TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Subayal Khan

System Level Performance Evaluation of Distributed Embedded Systems

Julkaisu 1099 • Publication 1099

Tampere 2012

Tampereen teknillinen yliopisto. Julkaisu 1099
Tampere University of Technology. Publication 1099

Subayal Khan

System Level Performance Evaluation of Distributed Embedded Systems

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB109, at Tampere University of Technology, on 30th of November 2012, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2012

ISBN 978-952-15-2969-6 (printed)
ISBN 978-952-15-3241-2 (PDF)
ISSN 1459-2045

Abstract

In order to evaluate the feasibility of the distributed embedded systems in different application domains at an early phase, the System Level Performance Evaluation (SLPE) must provide reliable estimates of the non-functional properties of the system such as end-to-end delays and packet losses rate. The values of these non-functional properties depend not only on the application layer of the OSI model but also on the technologies residing at the MAC, transport and Physical layers. Therefore, the system level performance evaluation methodology must provide functionally accurate models of the protocols and technologies operating at these layers.

After conducting a state of the art survey, it was found that the existing approaches for SLPE are either specialized for a particular domain of systems or apply a particular model of computation (MOC) for modelling the communication and synchronization between the different components of a distributed application. Therefore, these approaches abstract the functionalities of the data-link, Transport and MAC layers by the highly abstract message passing methods employed by the different models of computation. On the other hand, network simulators such as OMNeT++, ns-2 and Opnet do not provide the models for platform components of devices such as processors and memories and totally abstract the application processing by delays obtained via traffic generators. Therefore the system designer is not able to determine the potential impact of an application in terms of utilization of the platform used by the device. Hence, for a system level performance evaluation approach to estimate both the platform utilization and the non-functional properties which are a consequence of the lower layers of OSI models (such as end-to-end delays), it must provide the tools for automatic workload extraction of application workload models at various levels of refinement and functionally correct models of lower layers of OSI model (Transport MAC and Physical layers).

Since ABSOLUT is not restricted to a particular domain and also does not depend on any MOC, therefore it was selected for the extension to a system level performance evaluation approach for distributed embedded systems. The models of data-link and Transport layer protocols and automatic workload generation of system calls was not available in ABSOLUT performance evaluation methodology. The, thesis describes the design and modelling of these OSI model layers and automatic workload generation tool for system calls. The tools and models integrated to ABSOLUT methodology were used in a number of case studies.

The accuracy of the protocols was compared to network simulators and real systems. The results were 88% accurate for user space code of the application layer and provide an improvement of over 50% as compared to manual models for external libraries and system calls. The ABSOLUT physical layer models were found to be 99.8% accurate when compared to analytical models. The MAC and transport layer models were found to be 70-80% accurate when compared with the same scenarios simulated by ns-2 and OMNeT++ simulators. The bit error rates, frame error probability

and packet loss rates show close correlation with the analytical methods .i.e., over 99%, 92% and 80% respectively. Therefore the results of ABSOLUT framework for application layer outperform the results of performance evaluation approaches which employ virtual systems and at the same time provide as accurate estimates of the end-to-end delays and packet loss rate as network simulators. The results of the network simulators also vary in absolute values but they follow the same trend. Therefore, the extensions made to ABSOLUT allow the system designer to identify the potential bottlenecks in the system at different OSI model layers and evaluate the non-functional properties with a high level of accuracy. Also, if the system designer wants to focus entirely on the application layer, different models of computations can be easily instantiated on top of extended ABSOLUT framework to achieve higher simulation speeds as described in the thesis.

Acknowledgements

First of all I will like to thank all my colleagues, without their support and help, I won't be able to complete my doctoral studies in less than three years. I am really grateful to Research Prof. Juha-Pekka Soininen for defining challenging and interesting research problems for my doctoral thesis and Prof. Jari Nurmi for his continuous feedback, supervision and support. I am also grateful to Kyösti Rautiola for arranging a project for writing PHD thesis without which it won't be possible to complete the doctoral degree in less than three years. The frequent discussions with Juha-Pekka and his feedback and criticism on my work really made a huge difference. He has been very kind to discuss with me his ideas which helped me to steer the research in the right direction. I owe a lot to Prof. Jari Nurmi for arranging book exams which were related to my doctoral research. It was surely the best decision to choose Tampere University of Technology (TUT) for my doctoral studies. I will strongly recommend TUT to anyone who wants to pursue his doctoral studies in the area of Computer Systems, Software architectures and Distributed Systems.

Throughout my doctoral research, I worked hand in hand with Jukka Saastamoinen. His expertise and in depth knowledge in the diverse areas of platform architectures made the real difference due to which I progressed really fast in my doctoral studies. He has been really supportive and his personality has been a source of inspiration. The motivation and passion which he brings to his profession raised my motivation level which was important to complete the thesis in three years. Also many thanks for Tony Wong for his support. Both Jukka and Tony have inspired me a lot and I was so lucky to have worked with them during my doctoral studies.

I am really grateful to Jyrki Huusko and Mikko Majanen for providing me with the simulation models in ns-2 network simulator which played a key role in validation of MAC and Transport Layer models integrated to ABSOLUT. I am also grateful for their contribution to the research articles (many of which are a part of thesis) and valuable criticism on my modelled MAC and Transport Layer models. Without Mikko's OMNeT++ and ns-2 simulations it won't be possible to proceed since the functional corrections of my models would be doubtful. As a result of collaboration with Mikko and Jyrki, I have been able to model the functionally correct MAC protocols in ABSOLUT. The functional correctness of these protocols is reported after the simulation via specialized probes. Jukka, Tony, Jyrki and Mikko are amongst the best people and colleagues I ever met so far. I will be looking forward to work with them in future too.

I will also like to thank Kari Tiensyrjä for defining my yearly goals during development discussions. Also many thanks to Mätti and Jussi for helping me with the practical issues related to the Service and Information Level IOP platforms. I am also thankful to Jari Kreku for weekly discussions regarding the background studies related to embedded systems. He also provided me with case studies which helped me in learning SystemC very quickly. Also, Jari provided me with the results related to the accuracy of ABSINTH workload generation methodology which considerably reduced the time of my doctoral studies.

I am also grateful to Dr. Eila Ovaska and Susanna Pantsar-Syväniemi for their contributions to two research articles which acted as the foundation for my doctoral Thesis. The State of the Art Survey conducted together with Dr. Eila Ovaska helped me to identify the key research problems in the area of system level performance simulation. In the beginning I was a little hesitant to do that but now I realize the true value of conducting research in a disciplined manner. Susanna helped me to write my first research article and at that time I had no background knowledge of embedded and distributed systems. Her concrete background enabled me to grasp of the fundamental concepts very quickly. That won't have been possible without her help.

I will also like to thank Birgitta Henttunen, Sanna Downing , Eija Posio, Susanna Turpeinen, Tuija Soininen and Sari Mäläskä for helping me with the bureaucracy, travel plans claims and immigration issues. I must thank my friend Joonas since he has been very kind to join me in the badminton court which helped me learn a lot about Finnish culture.

In the end I will thank my sister, elder brother Junaid and his family and younger brother Tashfeen for their support. Unfortunately my mother passed away during by doctoral studies but her memories will always remain alive with me. I simply don't have the words to describe the role which she has played in my education and forging my personality.

In the end and most of all, many thanks to the Single God Almighty for giving me the energy, will power and burning desire to pursue the difficult goals with iron determination. In every impediment and challenge in life, when no one else was there, the unshakeable belief that you are around made the real difference and will make in future too.

Abbreviations

ABSINTH	ABStract INstruction exTraction Helper
ABSOLUT	Abstract Instruction Workload and Execution Platform Based Performance Simulation
ADIOS	Advanced Interoperability Solution
API	Application Programmers Interface
Artemis	Architectures and Methods for Embedded Media Systems
ARTS	Abstract system-level modelling and simulation framework
BER	Bit Error Rate
BSD	Berkeley Software Distributions
CAG	Communication Analysis Graph
CORRINA	COnfiguration and woRkload generatIon via code instrumeNtatioN and performAnce counters
CORRINA-PERFUME	CORRINA outPut parsER for FUnction workload generation and process ModEl configuration
CORRINA-SCENT	Source-Code parsEr written in pythoN for CORRINA Tags insertion
DFG	Data Flow Graph
DIP	Device Interconnect Protocol
ECU	Electronic Control Unit

EDP	Energy Delay Product
FER	Frame Error Rate
FSMD	Finite-State Machine with Datapath
GCC	GNU Compiler Collection
GENESYS	Generic Embedded System Platform
GenM	Generic Model
H_IN	Higher Interconnect
IOP	Interoperability
IPC	Inter Process Communication
ISS	Instruction Set Simulator
KP	Knowledge Processor
KPN	Kahn Process Network
L_IN	Lower Interconnect
MAC	Medium Access Control
MARTE	Modelling and Analysis of Real-Time and Embedded Systems
MESH	Modelling Environment for Hardware and Software
MIC	Model Integrated Computing
MILAN	Model-Based Integrated Simulation
MOC	Model of Computation
NFP	Non Functional Property

NoTA	Network on a Terminal Architecture
OCL	Object Constraint Language
OS	Operating System
OSCI	Open SystemC Initiative
OSI	Open Systems Interconnection
PER	Packet Error Rate
QoS	Quality of Service
RDF	Resource Description Framework
RIBS	RDF Information Base System
RTOS	Real-Time Operating System
SDL	System-Level Description Language
SESAME	Simulation of Embedded-System Architectures for Multi-Level Exploration
SIB	Semantic Information Broker
SLDL	System Level description Language
SLPE	System Level Performance Evaluation
SOA	Service Oriented Architecture
SOC	Systems on Chip
SPADE	System Level Performance Analysis and Design Space Exploration
TDEU	Trace Driven Execution Unit

TLM	Transaction Level Modelling
UML	Unified Modelling Language
WSN	Wireless Sensor Network

List of original publications

I Linking GENESYS application architecture modelling with platform performance simulation. Khan, Subayal; Pantsar-Syväniemi, Susanna; Kreku, Jari; Tiensyrjä, Kari; Soininen, Juha-Pekka. Forum on Specification and Design Languages 2009 (FDL2009). Sophia Antipolis, France, September 22-24, 2009. ECSI. France (2009)

II Performance evaluation of distributed applications via Kahn process networks and ABSOLUT. Khan, Subayal; Saastamoinen, Jukka; Huusko,Jyrki; Nurmi,Jari. The Fifth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies UBICOMM 2011. Lisbon, Nov. 20-25, 2011. IARIA (2011)

III Multi-threading support for system-level performance simulation of multi-core architectures. Saastamoinen Jukka; Khan, Subayal; Tiensyrjä, Kari; Taipale, Tapio. ARCS 2011. 24th International Conference on Architecture of Computing Systems 2011, Workshop Proceedings. VDE Verlag Gmbh (2011), 169-177

IV Analysing transport and MAC layer in system-level performance simulation .Khan, Subayal; Saastamoinen, Jukka; Majanen, Mikko; Huusko, Jyrki; Nurmi, Jari. 2011 International Symposium on System on Chip, SoC 2011. Tampere, Finland, 31 Oct. - 2 Nov. 2011. IEEE (2011).

V Khan, Subayal;Saastamoinen, Jukka; Jyrki, Huusko; Nurmi, Jari. Journal Article(2012a). “Application Workload Modelling via Run-Time Performance Statistics”. International Journal of Embedded and Real-Time Communication Systems (IJERTCS), 2012.

VI SLPE of distributed GENESYS applications on multi-core platforms. Khan, Subayal; Saastamoinen, Jukka; Tiensyrjä, Kari; Nurmi, Jari. The 9th IEEE international symposium on Embedded Computing (EmbeddedCom 2011). Sydney, Australia, Dec 12-14 2011.

VII Performance Evaluation of distributed NoTA applications on multi-core platforms. Khan, Subayal; Saastamoinen, Jukka; Nurmi, Jari. 2nd IEEE International Conference on Networked Embedded Systems for Enterprise Applications. NESEA 2011, Fremantle, Australia, Dec. 8-9, 2011. IEEE (2011)

Contents

Abstract ..... **II**

Acknowledgements ..... **IV**

Abbreviations ..... **VII**

List of original publications ..... **XI**

Contents ..... **XIII**

1. Introduction **1**

1.1 Problem Definition 3

1.2 Research Hypothesis 7

1.3 Research methods 10

1.4 Organization of Thesis and Author's contribution 11

2. Survey of Design Space Exploration Methodologies **14**

2.1 Key concepts 14

2.1.1 Architectural Exploration 14

2.1.2 Y-Chart 14

2.1.3 Platform Model 15

2.1.4 Application Model 16

2.1.5 Mapping and Co-Simulation 16

2.1.6 Model of Computation (MOC) 17

2.1.7 Instruction set simulator (ISS) 17

2.2 Landmark Methodologies 18

2.2.1 Artemis (KNP-MOC) 18

2.2.2 ARTS 18

2.2.3 Baghdadi et al 19

2.2.4 Fornaciari et al 19

2.2.5 Jabaer et al 20

2.2.6 Koski 20

2.2.7 Lahiri et al 21

2.2.8 MESH 21
2.2.9 MILAN 22
2.2.10 Posadas et al 22
2.2.11 ReSP 23
2.2.12 SPADE 23
2.2.13 StepNP 23
2.2.14 TAPES 24
2.3 Overview of ABSOLUT 25
2.3.1 Structure of Application Workload Models 25
2.3.2 Application workload modelling techniques 26
2.3.3 Platform Modelling 27
2.3.4 Mapping and co-simulation 28
2.4 Comparison of Methods and Tools 28
2.4.1 Modelling Style 29
2.4.2 Languages, Standards and Frameworks 30
2.4.3 Non-functional properties validation 30
2.4.4 Targeted systems Domain 31
2.5 Summary 37
3. Towards Performance Evaluation of distributed systems 39
3.1 Requirements for SLPE of distributed systems 39
3.2 Feasibility of Existing SLPE Approaches 41
3.3 Enhancements needed for ABSOLUT 42
3.4 Summary 45
4. Multi-threading support for SLPE 46
4.1 System synchronization 46
4.2 Inter-process communication and system-synchronization model 48
4.3 Summary 50
5. Modelling OS Services (OS_Services) and Background Processes 51
5.1 Deriving new OS_Services 52
5.2 Registration and Access of OS services 53

5.3 Modelling of Background Processes 54
5.4 Data-link and Transport Layer Services 56
5.5 M3 Service and Information Level Services 57
5.5.1 M3 Service Level Models 59
5.5.2 SLPE of M3 applications 61
5.6 Summary 63
6. Instantiating KPN MOC over ABSOLUT 65
6.1 Properties of KPN MOC 65
a. Read/Write Operations to channels. 65
b. FIFO channel read/write operations 65
c. Channel Access 65
d. Process code behaviour 65
e. FIFO channels 65
f. Definition of a KPN processes in ABSOLUT context 65
6.2 Implementing KPN Message Passing Services 66
6.3 KPN FIFO Channels and Token Modelling 70
6.4 Token Passing and Reception 70
6.5 A KPN process in ABSOLUT context 71
6.6 Summary 74
7. Workload Modelling via Run-Time performance statistics 75
7.1 Methodology 75
7.2 Comparing ABSINTH, ABSINTH-2 and CORRINA 78
7.3 Overheads of CORRINA 79
7.4 Summary 80
8. Modelling of Performance probes 81
8.1 Modelling of Probes for NoTA systems 81
8.2 Measuring Delays and processing times 82
8.3 Related Case Studies 83
9. Accuracy of Models/Tools 84
9.1 Software by composition 84

9.2 ABSOLUT Models/Tools and real systems 86
9.3 Accuracy of Physical Layer models 87
9.3.1 Comparing Bit error rate (BER) with analytical models. 88
9.3.2 Comparing Frame error rate (FER) with analytical models 88
9.3.3 Comparing PER with analytical models 89
9.4 Accuracy of System Call Models (OS_Services) 91
9.4.1 End-to-end delays and throughput 91
9.4.2 Platform utilization and Contribution to Overall Workload 97
9.5 Accuracy of Application Workload models 100
9.5.1 Accuracy of User Space code Workload Models 101
9.5.2 Accuracy of External Libraries workload models 102
9.6 Middleware and Higher Level Protocols 103
9.7 Summary 108
10. Introduction to papers 109
11. Conclusions and discussion 111
References 114

1. Introduction

An embedded system can be defined as a special-purpose computing system (meant for information processing) which is closely integrated into the environment. An embedded system is generally dedicated to a particular application domain. Therefore, the embedding into a technical environment and the constraints that are a consequence of their application domain mostly result in implementations that are both heterogeneous and distributed. In such cases, the systems comprise of hardware components which communicate by means of an interconnection network (Simon Perathoner et al., 2008).

Due to the dedication to a particular application domain, heterogeneous distributed implementations are common. In such implementations, each node specializes by incorporating communication protocols and other functionalities which facilitate optimum and reliable performance in its local environment (Simon Perathoner et al., 2008). For example, in automotive applications, each network node (usually called embedded control units), contains a communication controller, a CPU, memory, and I/O interfaces (Simon Perathoner et al., 2008). But as per functionality, a particular node in the network might contain additional hardware resources such as digital signal processors, CPUs and different memory capacity (Simon Perathoner et al., 2008).

Distributed embedded systems can be classified as real-time and non-real-time distributed systems. The real-time systems are required to complete their tasks or deliver their services within a certain time frame. In other words, the real-time systems have strict timing requirements which must be met. Digital control, signal processing and telecommunication systems (Insup Lee ,2007) are usually distributed real-time systems. On the other hand, PCs and workstations which run non-real-time applications, such as our email clients, text editors and network browsers are common examples of distributed non-real time systems.

Also, the networking revolution is driving an efflorescence of new distributed systems for new application domains for example tele-surgery, smart cars, unmanned air vehicles and autonomous underwater vehicles. The components of these systems are distributed over the Internet or wireless LANs (Insup Lee ,2007). Due to these technological advancements, the spatial limitations seem to be progressively fading away which has given rise to new paradigms such as mobile computing (G. Forman and J. Zahorjan, 1994). These technologies have enabled us to connect to the Internet while we are on the move via pocket-sized, battery-powered embedded devices for example PDAs (personal digital assistants) and cellular phones which communicate over a wireless channel. The applications of computing devices have also been changing in response to ever-improving hardware and software technologies (Insup Lee ,2007). Nowadays we routinely use a variety of multimedia-based services instead of text-based ones by using nomadic hand-held devices such as high-end mobile phones.

The burgeoning market for information, entertainment, and other content-rich services can be seen as a consequence of the rising popularity of mobile devices with high-quality multimedia capabilities (K. Tachikawa, 2003) .These services should not just adapt to a continuously changing computing environment but also meet the different requirements of individual users. In such cases, we need to consider additional real-time and embedded non-functional properties of multimedia applications for example, the maximum allowable time for each delivered packet and battery life. Therefore,

the challenge from the system design perspective is to reduce the form factor and energy consumption of the mobile nomadic devices, thus increasing the portability and durability of these devices. This will enable these devices to be used by many customers for everyday use by maintaining low power consumption, which is important due to limited power available from the battery. Since the Moore's law predicts that the computing power will continue to increase, the energy constraints demand that we shall sacrifice the performance in portable devices in return for a longer operation time. Generally speaking, the focus of recent research has been the topics such as efficient usage of storage space, various I/O devices (A. Vahdat, 2000), (E. Pitoura, 1999), (M.Weiser, 1994) and processing elements available from the device platforms.

Wireless sensor networks (WSNs) are also an example of distributed real-time embedded systems which are composed of a cooperative network of nodes (J. Hill et al., 2000). Due to small form factor of the network nodes, each consists of limited processing capability (for example microcontrollers, CPUs, or DSP chips) and memory (program, data, and flash memories) resources. Each node has an RF transceiver, a power source (e.g., batteries and solar cells), and contains sensors and/or actuators. The nodes communicate wirelessly and have the ability to self-organize after adhoc deployment. WSNs of 1,000s or even 10,000 nodes are anticipated and are perceived to revolutionize the way we live and work. Since WSNs are distributed real-time systems which are rapidly evolving technologically, an important question is to know that: how many existing solutions (Transport protocols and Data-link protocols etc.,) for existing distributed and real-time systems can be used in these systems? It has become obvious that many protocols which were developed beforehand would not perform well in the domain of WSNs. The reason is that the WSNs do not employ many of the assumptions underlying the previous networks for example Medium Access Control (MAC) protocols.

A MAC protocol is employed by the network nodes for the coordination of actions over a shared channel. The most commonly used MAC protocols are contention-based. One generally used distributed contention-based strategy is that a node which has a message to transmit, tests the channel to see if it is busy, if not busy then it transmits, if the channel is busy, it waits and tries again later. In most cases, MAC protocols are optimized for the general cases and arbitrary communication patterns and workloads. Contrarily, WSNs have more specific requirements which include a local unicast or broadcast. The traffic flow is usually from many nodes towards one or a few sinks (most traffic is thus directed in one direction). The individual nodes have periodic or rare communication and must consider energy consumption as a major factor. An effective MAC protocol for WSNs must have reduced power consumption, shall avoid collisions, should be implemented with a small code size and memory requirements, be efficient for a single application and be tolerant to changing radio frequency and networking conditions (Insup Lee , 2007). That is why many WSNs employ highly efficient MAC protocols for the transfer of frames over the wireless channels for example NANO MAC (J. Haapola, 2003) AND BMAC (J.Polastre, 2004).

In order to manage the increasing complexity and heterogeneity of the distributed embedded and computer systems, middleware technologies such as CORBA (Steve Vinoski , 1997) and NoTA (Khan et al. , 2011c) are becoming increasingly important. A middleware can be viewed as a system layer implemented as software which can perform tasks such as mobility management, connectivity, and resource management. Middleware solutions break the complexity of a system via loosely con-

nected services on top of a number of heterogeneous sub-systems (Steve Vinoski , 1997). The advanced middleware technologies have become the driving force of innovation in many area of distributed embedded and computer systems.

For many of these applications, information sharing among the various distributed components is critical. In many cases the devices running the distributed application components constitute smart spaces and the information sharing among them requires semantic level interoperability (J. Kiljander et al. 2011). The goal of semantic level interoperability is to enable the meaningful sharing of information between varieties of devices. M3 is a concept for making use of the Semantic Web ideas and technologies for providing the semantic level interoperability between devices in physical environments. For the industrial deployment of these novel paradigms, the feasibility of application employing these solutions must be evaluated from various aspects such as the architecture of the solution and the complexity of the application components. Therefore, from the aforementioned discussion, it is obvious that the main pillars of advancement and innovation in the area of Information and Communication Technologies (ICT) stand on the ground of novel applications of distributed embedded systems.

Based on the assessment of the current chapter, it is obvious that the distributed embedded systems are inherently difficult to design and to analyse. In many situations, not only the availability, safety, and the correctness of computations of the whole embedded system are major concerns but also the timeliness of the results (Simon Perathoner *et al.,* 2008).

It is therefore evident that the complexity of the distributed systems has increased enormously in almost all industrial domains which they span and are thus accompanied with various design challenges. Firstly, the system design space is huge not only due to many alternatives for data-link, transport and middleware technologies (for example specialized MAC protocols in WSNs and middleware technologies in multimedia applications domain) but also in terms of available platforms and application implementation choices. Secondly, due to computational complexity of many distributed applications and the design constraints in terms of non-functional properties, the designer has to make critical design decisions at an early stage in order to evaluate a particular system design with other possible alternatives before the actual implementation and integration of the system starts. Moreover, both the functional and non-functional properties of the overall distributed system not just depend on the computations performed within the network nodes but also on the interaction of the various data streams on the common communication media. In contrast to both the multiprocessor and parallel computing platforms, each computing node within the network has a high degree of independence and usually communicates with other node(s) via message passing. It is particularly difficult to maintain not only the global state but also the workload information since the processing nodes usually make independent scheduling and resource access decisions (Insup Lee, 2007).

1.1 Problem Definition

Designing, implementing, and evaluating distributed embedded systems of different domains bring several challenges, including Quality of Service (QoS) support, component integration, and configuration management. As a result, the software is increasingly distributed onto networks and structured into logical components that interact asynchronously.

The rapid innovations in radio, transport and middleware technologies have resulted in many new application areas of distributed computer systems. Just two decades back, no one could even imagine that traffic lights will be optimized via wireless WSNs (Sensinode) for energy reduction, mobile phones will play the content rich multimedia applications and the functionalities of decade old robust desktop computers will be outperformed in many respects by the high-end portable devices of today like iPad, iPhone and high-end Android mobile phones. All this is due to the intricate technologies which reside at different layers (Network, Data link and Physical layers) of the Open Systems Interconnection (OSI) model. Due to these developments, the overall complexity of the distributed embedded systems has increased tremendously over the last two decades. Hence the system designer must evaluate the feasibility of different technologies at different layers of the OSI model as well as possibly middleware technologies. This will result in early design decisions which will gradually shrink the design space and result in a robust and optimum distributed system.

Due to the convergence of computer systems and software engineering, model-based development of embedded systems has surfaced as one of the most profound trends in technology today. System designers build new, robust and adaptive distributed systems via computing and communication technologies. Also, large-scale systems of networked embedded subsystems can be integrated for various applications. Software engineers on the other hand implement software to satisfy the functional and non-functional requirements that are simultaneously physical and computational. The models employing different abstraction levels have been widely used in the design of distributed systems and in the domain of software engineering (Insup Lee ,2007).

Gradually, more and more computer systems have been embedded into the physical environments. It is expected that this trend will continue in order to improve our quality of life. In real-time distributed embedded applications, the end-to-end real-time performance is critical to ensure emergency response within bounded delay. It is important to note that timeliness is a property which is tightly related to the management of shared resources (e.g., wireless medium). The challenges in the design of medium access control (MAC) layer protocols arise in many domains of distributed embedded systems such as WSNs. That is why, in case of WSNs, the networked devices use lightweight and delay sensitive network-layer MAC protocol (J.Haapola, 2003). Such protocols usually estimate the network capacity to quantify the ability of the network to transmit information in time.

In order to abstract issues of distribution, heterogeneity, and programming language from the design of systems, middleware is being applied to many distributed systems (Insup Lee, 2007). There are many types of middleware technologies for example context aware middleware gathers the information related to the context in which distributed applications are operating and makes it available to the components of the distributed software components. Also, some middleware technologies are targeted for adhoc systems and provide automated Service Discovery and QOS management while others improve hardware (hw) component modularity by applying ideas from Service Oriented Architecture (SOA) with the help of an interconnect (which is a network). One example of middleware technology is Network on a Terminal Architecture (NoTA), which is intended to be device-internal system for supporting the rapid development of hw/software(sw) systems based on modular components. A NoTA device may or may not support ad hoc networking (Khan et al. , 2011c). NoTA uses IP multicast for node discovery. QoS might be provided through the interfaces and is im-

plementation specific. NoTA was an interesting industry effort to solve a very important problem; however, it appears that the development is not very active at the moment.

Therefore, the design of distributed systems is challenging due to their diverse application areas. The use of general purpose or specialized transport, data-link and middleware technologies depends on the application domain. Hence, the design of distributed systems can be viewed as a sequence of possible design decisions related to the selection (as per application domain) of appropriate technologies operating at different layers of the OSI model and also the selection of proper middleware technologies (if they are used). For example in case of WSNs, it is very important to use a highly specialized MAC protocol at data-link layer such as nanoMAC (J.Haapola, 2003). Hence, depending on the application domain, at first, different alternatives at each layer of the OSI model are evaluated and the most suitable ones are selected for the implementation phase.

After identifying the appropriate communication/middleware technologies, the platform model comprising the models of selected protocols is instantiated which is then tested with different application design alternatives. Depending on the use-case, in most cases, the platform with the lowest computational power which satisfies the non-functional properties is selected. Also, some applications, for example online games for high-end mobile phones, are developed to run on a variety of platforms. In these cases, the application might also be implemented in a variety of ways in order to ensure good end-user experience on different platforms. Therefore, while designing a distributed computer system, the system designer can possibly consider many alternatives of transport, MAC protocols, middleware technologies, application implementations and platforms as shown in

Figure **1**.

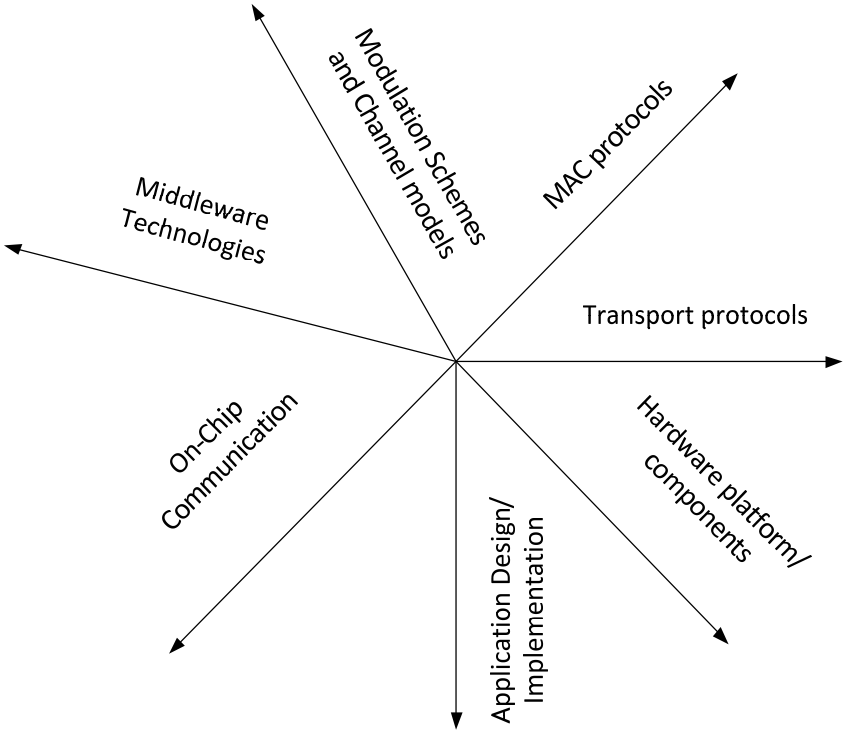

Figure 1: Dimensions (alternatives of a specific technology) considered for the design of distributed embedded systems

We therefore conclude that the complexity of the distributed embedded systems in many application domains is huge. In order to simplify the design of these complex systems, the methodology used for architectural exploration must provide the tools and models which will enable efficient design space exploration by validating the non-functional properties of the system. The methodology should also report the contribution of protocols at different layers of the OSI model as well as middleware technologies in non-functional properties such as end-to-end delays. The early phase performance simulation of the distributed computer systems must use functionally accurate models of MAC protocols, transport protocols and middleware technologies since, as stated before, in case of distributed applications contribute to the non-functional properties.

The non-functional properties (NFPs) must be explicitly mentioned in the application views and carried through the app design phase. These NFPs must be validated by the performance simulation phase before the system deployment starts. The application workload models must mimic the workload and control of the application. Also, the platform models must be fast enough to allow faster simulation speed. This will lead to faster iterations in architectural exploration phase to ensure faster progress to the system development and thus faster time to market. In this way, the launched

product will be able to make full use of the market window. Therefore, the cycle accurate platform component models cannot be used, and instead, cycle approximate models are mostly used. The methodology should not be restricted to a certain domain of applications. Many existing performance simulation approaches employ Kahn Process Network (KPN) Model of Computation (MOC) and can therefore model only the streaming multimedia applications well (Lieverse , 2001a,b) (Zivkovic, 2002). Also, the approach must provide a library of platform components for instantiating a variety of platform models. The methodology must also use standard tools, freely available libraries, free yet standard simulation environments and modelling standards for designing new platform models for brisk industrial deployment. Apart from that, the methodology must provide a sufficient level of automation for the extraction of application workload models from application source code (if available) (Kreku *et al*., 2008a, b). The methodology must also demonstrate the way application workload models can be derived from the application model if the application has not been implemented (Kreku *et al*., 2008a, b).

In order for the methodology to scale to different domains of both real-time and non-real time distributed systems, it must provide a framework for deriving new protocol models. These models must incorporate sufficient functionality of the modelled protocols/technologies so that system designer can take important design decisions to steer the trajectory towards a more optimal system design (Kienhuis, 1997). The tools and methods used for design space exploration must provide the system designer with a comprehensive post-simulation report of NFPs after every iteration in the design space exploration. If after thorough analysis of the report, the system designer observes that the NFPs of the system have been fulfilled, the system development starts. If the NFPs are not satisfied, iteration proceeds with modified application or platform model.

1.2 Research Hypothesis

The research was conducted with an aim to develop the models and tools for early phase System Level Performance Evaluation (SLPE) of distributed embedded systems. The goal was to provide functionally correct models of MAC and transport protocols as a part of a system level performance evaluation methodology. Apart from that, the methodology must be extensible so as to provide the ability to integrate newly developed MAC and transport protocols into the device platform models. The models must reflect the layered design of the OSI protocol stack, for example the higher layer protocol models employed by the methodology should use the lower layer protocols.

For example, the transport protocol models should use the MAC protocol models in the same way as in OSI model. Also the performance modelling methodology must be capable of extracting the workload models of middleware technologies. This is important since the middleware technologies are gaining popularity due to the increasing complexity of the distributed applications and the rapid proliferation of SOA based approaches for application design. The performance modelling approach must provide a library of components for instantiating platform models of embedded devices involved in the distributed embedded system. Also, the workload models must be generated automatically to reduce the time and effort involved in performance evaluation. These research problems can be best described in terms of the following questions.

1. Can the OSI Model layers for example transport and MAC be successfully designed and integrated to a SLPE design Methodology while retaining sufficient functionality so that their contribution to non-functional properties is estimated with a high degree of accuracy?

2. Can the Protocol models operating at different layers of the OSI model use the lower layer models in the same way as the layered OSI model (so as to build a wide variety of upper layer protocols models on top of lower layer protocol models)?

3. Can the application workload models used in the performance simulation phase truly represent the structure of the application model components? In other words can the application model act as a blue print for the performance simulation phase?

4. Can the methodology be successfully applied for the performance evaluation of applications designed via novel application/system design approached such as Network on a Terminal Architecture (NoTA) (Khan et al., 2011c) and Generic Embedded System Platform (GENESYS) (Khan et al., 2009)?

5. How efficient it is to instantiate the novel application design methodologies with UML 2.0 modern application design profiles such as Unified Modelling Language 2.0 (UML 2.0) Modelling and Analysis of Real-Time and Embedded Systems (MARTE) profile and how easy it is to validate the non-functional properties modelled and carried through the application design phase?

6. Can the delays be measured and reported at different layers of the OSI models to evaluate their feasibility in various use-cases and different domains of distributed systems such as WSNs?

7. Is it possible to evaluate the feasibility of MAC and transport protocols in isolation? This will be useful in distributed systems domains such as WSNs which employ specialized MAC protocols due to strict energy constraints.

Therefore the system level performance evaluation of distributed systems is challenging and the performance models must contain the models of different OSI model layers and must provide means to evaluate not just the performance of applications and hardware components of the devices which are part of the distributed system but also the MAC and transport protocols etc., . This is important since in case of distributed systems, the end-user experience can be affected by non-functional properties such as end-to-end delays. These non-functional properties can only be estimated with high accuracy if the functionally correct MAC and transport protocols are used in the per-

formance models. Therefore, it is important to integrate functional MAC and Transport protocols to the employed system-level performance evaluation approach. It will allow the system designer to estimate the non-functional properties such as end to end transport and MAC delays apart from the platform component usage/utilization as in the traditional methods.

The hypothesis in distributed embedded systems research is that it is possible to estimate the potential bottlenecks at different layers in the OSI model as well as the application implementation which will results in more optimal final distributed system design. It is important to know in many distributed embedded systems domains that to which extent the MAC and transport protocols or middleware technologies contribute in terms of platform usage in proportion to the overall cost of the application. This will allow the system designers to integrate the best possible MAC and transport protocols which can support a particular use-case or distributed systems domain. Also, if the communication between processes running on different embedded devices is modelled by abstracting out the Transport and MAC layers, the non-functional properties such as Frame and Packet delays will be inaccurate since they won't be a consequence of functionality at these layers. This is one of the major shortcomings of the Kahn Process Network (KPN) Model of Computation (MOC) based approaches which use FIFO channels for modelling communications between processes.

From event driven simulations perspective, the hypothesis is that is it possible to implement the functionally correct MAC protocols such as IEEE 802.11 Distributed Coordination Function (DCF) over the SystemC's model of computation (MOC). The modelled protocols can produce results which follow the same trend as widely used network simulators such as OMNeT++ and ns-2.

From the system level performance evaluation perspective, the hypothesis is that it is possible to model the application workloads at different level of refinement and detail so that the system designer can freely choose the workload modelling methodology which provides the desired balance between accuracy and speed. One novel application workload modelling tool-chain has been designed and developed which is based on run-time performance statistics and is summarized in Chapter 7 which allows the system designer to automatically model the workloads of the system calls and external libraries.

The novelty of the proposed approach stems from the fact that the modelled MAC, transport and middleware protocol/stack models are accurate enough for the SLPE as described in Chapter 9. The application models employed by the methodology act as a blue print for the application workload models used in the performance models. This reduces the time and effort in the performance simulation phase and shorter time to market. The results related to application execution times and usage of different platform components can be obtained to perform necessary optimizations (Khan et. al., 2009, 2011(a, c, d)). Also, the contributions of the MAC and Transport protocols in the end-to-end delays are reported separately. Traffic generators can be used for extremely high speed analysis of MAC, transport and middleware technologies in isolation. This will result in the selection of most suited protocols for the distributed embedded system under design. Also, the application workload models can be automatically generated by using the provided tools at different levels of refinement and detail which allows the system designer to decide the right balance between speed and accuracy. Furthermore, the non-functional properties in the application model represented in the application model are validated by the performance simulation phase.

1.3 Research methods

The main emphasis of the research presented in this thesis is the design and implementation of tools and models for the system level performance simulation of distributed embedded systems.

The research was conducted in different discrete phases. In the first phase, a state of the art survey of the existing SLPE approaches was conducted to evaluate their feasibility and shortcomings for the performance evaluation of distributed systems. The design of the distributed systems starts off by defining a number of non-functional properties. The allowable (range of) values of these non-functional properties are decided by the system designer after analysing the customer expectations. In case of distributed embedded systems these non-functional properties include end-to-end delays and packet losses etc., apart from platform utilization by the application(s) involved in the use of case. The components of the distributed applications run on different devices. The first phase provided valuable insight into the different aspects of the salient SLPE approaches such as modelling style, languages and tools, non-functional properties validation and targeted systems domain as listed in Chapter 2.

In the second phase, the main shortcomings of the existing SLPE approaches for performance modelling and non-functional properties validation of distributed systems were identified. The validation of non-functional properties of distributed systems requires the SLPE approach to provide functional models of MAC, Transport and other layers of the OSI model. It was found that none of SLPE approaches provides such models. Furthermore, many approaches are based on a certain Model of Computation (MOC) which makes it very inefficient and difficult to model the protocols/technologies at different layers of OSI model. Also on the other hand, some are particularly designed for a particular domain of computer systems. ABSOLUT was selected as the candidate for extension to SLPE of distributed systems since it does not employ any MOC and also it is not restricted to any particular domain of computer systems.

In the third phase, the shortcomings of ABSOLUT such as the lack of multi-threading application modelling support, functionally correct OSI model layers (MAC, Transport and Physical layer models) and lack of automatic workload generation of system calls were identified. The second and third phase is covered in Chapter 3 of the thesis.

In the fourth phase, the aforementioned OSI layers models were designed as Operating system(OS) services and integrated to the ABSOLUT framework. These models include Transport layer services, MAC layer services, Physical layer services and the workload models for middleware technologies. Apart from that the inability of ABSOLUT framework to automatically generate the workload models for system Calls was fulfilled by the design and implementation of a workload generation methodology based on run-time performance statistic called CORRINA. These extensions also allow the instantiation of any particular model of computation over ABSOLUT for example Kahn Process Network (KPN) MOC. This allows the system designer to evaluate certain systems belonging to a particular domain faster by abstracting the functionalities of middle-ware, Transport and lower layers of OSI model. This phase spans Chapters 4,5,6,7,8 and 9 of the thesis. The tools and models developed during the research were used in several extensively case studies. Chapter 9 sole-

ly focuses on the accuracy of the modelled protocols and tools while the other focus on the design and implementation of the models and tools.

1.4 Organization of Thesis and Author's contribution

The thesis comprises of an introductory part and VII original papers. An overview of these papers is provided in Chapter 10. Out of these VII papers, I was the first author of all the research articles except paper III. However my contribution in this research article was essential. These papers were a result of the work which I conducted between April-2009 and September 2011. The work was performed in the Artemis SOFIA project partially funded by the Finnish Funding Agency for Technology and Innovation (TEKES) and the European Union.

Chapter 2 provides a survey of the landmark approaches in the area of SLPE and challenges in the system level performance simulation of distributed systems. The Chapter describes the different embedded systems domains, modelling tools and languages spanned by these approaches. Also it illustrates the model of computations and other salient features of these approaches. The chapter also provides an overview of ABSOLUT and compares the different approaches on the basis of the modelling styles, tools and languages used and feasibility in terms of validation of non-functional properties of the distributed embedded systems at an early stage. Chapter 3 is based on the conclusions drawn in Chapter 2. First of all, the chapter lists the requirements for SLPE of distribute embedded systems and later on evaluates the feasibility of salient SLPE approaches discussed in Chapter 2 for performance evaluation of distributed embedded systems. The chapter first shortlists the salient approaches which are not restricted to the performance simulation of particular domains of embedded/computer systems. Afterwards, the chapter further shortlists the selected methodologies on the basis of application modelling without employing a MOC. The usage of a particular MOC for application modelling restricts the methodology to a particular domain of applications/systems. Therefore, such methodologies cannot be extended/used for the development of a general purpose and extensible framework for SLPE of distributed embedded systems which belong to different domains. The literature review reveals that out of all the landmark SLPE approaches described in the survey, only ABSOLUT has the potential to be extended for the development of such a framework. Of course, all the SLPE approaches were not covered in the survey. The survey acts as a disciplined approach for knowing the potential of a certain methodology for the performance evaluation and validation of non-functional properties of distributed computer systems.

Chapter 4 describes the way multithreading support has been integrated to ABSOLUT. Chapter 5 Chapter 6 and Chapter 7 describe the design and integration of communication technologies models, middleware technologies models and non-compiler based application workload extraction methods to ABSOLUT framework. Chapter 8 describes the way performance probes are modelled for recording (during simulation) and reporting (after simulation) the simulation results. An overview of the author's contribution described in these chapters is presented in Figure 2.

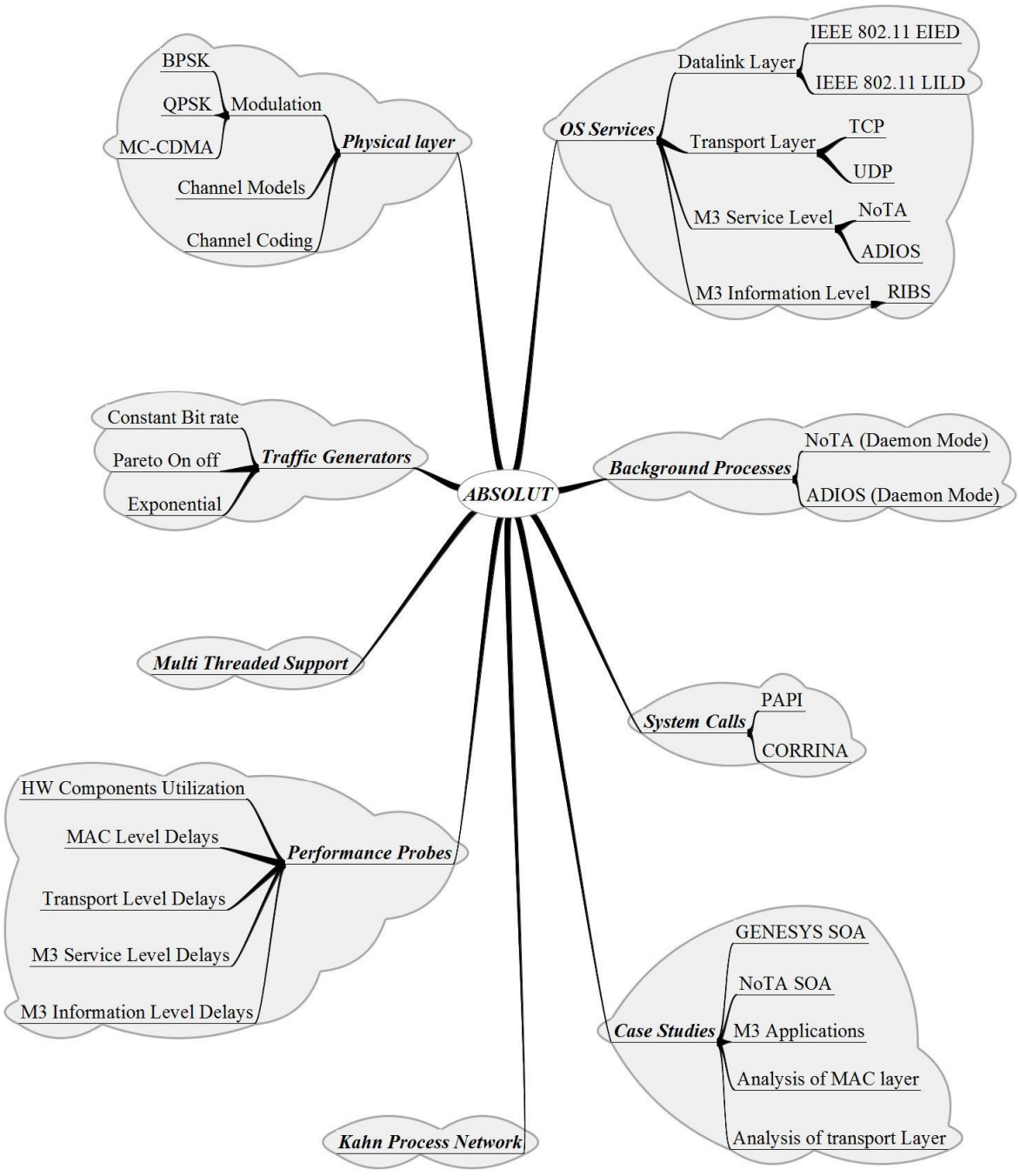

Figure 2: Overview of the Author's Contribution

As shown in Figure 2, the thesis also described the modelling of background processes and instantiation of KPN MOC (over ABSOLUT to model communication between processes of ABSOLUT application models). The models of background processes are important since many real world applications avail the functionalities provided by background processes. Also, many middleware technologies such as NoTA DIP can operate as a background process also (daemon mode). KPN MOC is useful for performance modelling of use cases where the contribution of Transport and MAC layer in the system's non-functional properties (for example end-to-end packet delays) and performance (platform utilization) is negligible. As described in Chapter 2, many SLPE approaches use KPN MOC for modelling streaming multimedia applications. Chapter 9 describes the accuracy of the modelled protocols and workload generation methods. Chapter 10 describes the way research contributions presented in different chapters are related to the original research articles. Chapter 11 focuses on the conclusions which are followed by the list of References.

2. Survey of Design Space Exploration Methodologies

In this chapter, we first provide an overview of the landmark contributions in the area of SLPE and ABSOLUT. Afterwards, we compare ABSOLUT to the salient performance evaluation approaches. The main concepts in the area of SLPE are described first; this is followed by an overview of the ABSOLUT. The comparison provided in this chapter is used to evaluate the feasibility of these approaches for the SLPE of distributed embedded systems in Chapter 3.

2.1 Key concepts

In this section, the common concepts that span different SLPE approaches are discussed. These concepts form the core principles of the landmark techniques developed in this area.

2.1.1 Architectural Exploration

Since the complexity of the modern embedded systems has increased considerably, it is not uncommon to recognize the impact of designer's experience on the final system design. Therefore, the design decisions taken at the early stages of the system development might be biased by the prior projects completed by the system design team. This effect can be observed in different application domains, for example in the application domain of network processing we see multitude of architectures which implement the same kind of application (N. Shah, 2001). The reason obviously is the dominant role of decisions which are a result of the prior experience of design team rather than the decisions resulting from the application-driven architecture design. It implies that for a given specification of the system and application requirements, the design team shortlists the design alternatives which are similar to earlier designs. Such designs might be sub-optimal for the current design problem. It means that the overall complexity of the distributed embedded systems is increasing due to development of various specialized MAC protocols, transport protocols and middleware technologies for particular application domains. As a result, the probability of sub-optimal system designs is also increasing as due to ad-hoc system design approaches. The design of distributed embedded systems is also becoming more challenging due to increasing use of heterogeneous architectures. Such architectures are mostly equipped with both general purpose and application specific computing resources. In order to make unbiased design decision at early stages of the system design and to handle design complexity, exploration tasks are mostly performed on the system level.

2.1.2 Y-Chart

The Y-chart (Kienhuis, 1997) is an approach for design space exploration which segregates the platform simulation model and applications performance models as shown in Figure 3.

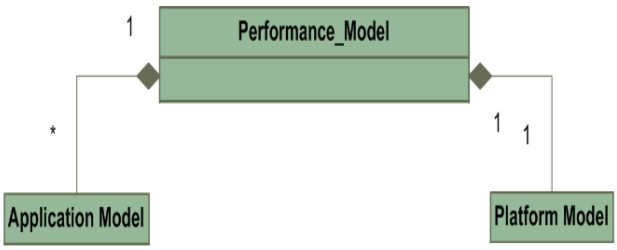

Figure 3: Main parts of performance model

The application model is mapped to the corresponding platform model and co-simulated as shown in Figure 4.

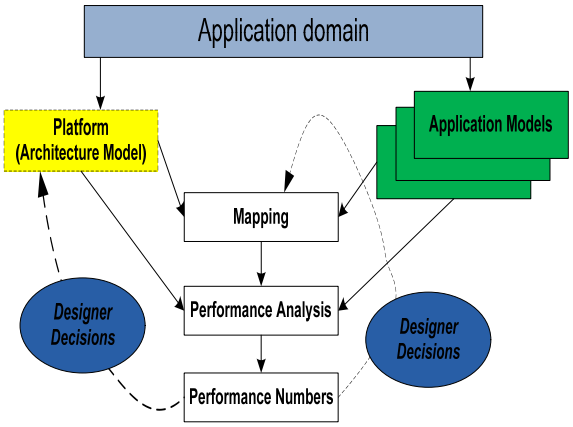

Figure 4: The Y-Chart

The performance numbers are investigated by the designer and if performance numbers comply with target performance requirements, architectural exploration ends and development phase starts. If the performance requirements are not met, the platform model or the application model or both are updated. Such iterations might occur several times during architectural exploration.

2.1.3 Platform Model

All the landmark SLPE approaches used for performance evaluation model the platforms at higher abstraction levels. Some SLPE approach use higher level transaction level models that emphasize on separating computation from communication and some model the platform architecture by means of a graph. Furthermore, KPNs have also been extensively employed for the platform modeling as shown in Figure 5.

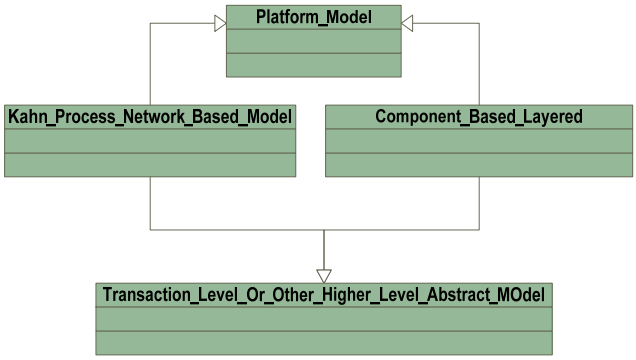

Figure 5: Platform models used for performance evaluation

2.1.4 Application Model

For performance simulation, the complete functional specifications are not executed at simulation run-time. Instead, a situation is reproduced which mimics the actual execution of the application on the system architecture. Performance models don't have to be complete and unambiguous. Some performance evaluation techniques adopt a transaction level approach to model applications. Traces have also been used to model the behavior of the applications at a high abstraction level and their interaction on the architecture. In this way, the need for a fully functional application model during architecture exploration is avoided. Application models usually use KPN MOC or follow a layered modeling approach as shown in Figure 6.

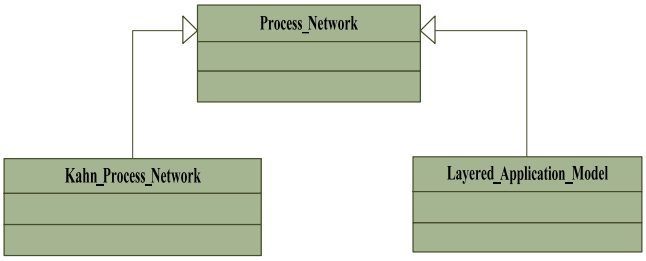

Figure 6: Application models used for performance evaluation

2.1.5 Mapping and Co-Simulation

Performance evaluation strategies demand the recording of data used by computation and communication resources and identification of bottlenecks in the system. After defining the application and architecture model, the next step is to map application subtasks to architectural resources. The overall performance model is co-simulated and when the simulation exits, the performance number are evaluated by the system designer to perform necessary changes to the platform or the application model.

2.1.6 Model of Computation (MOC)

A MOC is a general way of describing the behavior of a system in an abstract and conceptual form. In general, the MOC is described in a formal manner via mathematical functions or set-theoretical notations or a combination of them (Khan et al., 2011a). MOCs are abstract representations of computer systems and deal with a set of theoretical choices on the top of which the execution models of the computer languages are build. A MOC therefore provides an abstract paradigm of visualization with a particular goal which can be instantiated on a variety of systems.

Kahn Process networks (KPNs)

KPNs (Khan et al., 2011a) consist of nodes (which represent processes) and arcs which represent communication channels between the nodes. In order to model parallel computation, some autonomous computing nodes are connected to each other in a network by communication lines. A given node performs computation on the incoming data via the input lines using some memory of its own, to produce output on some or all of its output lines (Khan et al., 2011a). A communication line (an unbounded FIFO channel) transmits information within an unpredictable and finite time (Khan et al., 2011a). At any time a node either computes or waits for information on one of its input lines (Khan et al., 2011a). Each node follows a sequential program (Khan et al., 2011a).

Transaction Level Modelling (TLM)

TLM (Weiwei Chen, 2009) is a widely used approach to model digital systems. It is mostly used in those cases where the details of communication among system entities are abstracted out. In fact, TLM does not provide the specifications which would converge to a well-defined MOC and instead relies on the system design flow and employs system level description language (SLDL) for defining the details of supported syntax and semantics.

Data Flow Graphs (DFGs)

DFGs and the MOCs derived from it are widely used for describing computational intensive systems (Weiwei Chen, 2009). When a DFG MOC it is combined with Finite State Machine (FSM) (which is mostly used for describing control systems), the combination of both (FSM and DFG) results in Finite-State Machine with Datapath (FSMD). FSMD is used to describe systems which require both control and computation (Weiwei Chen, 2009).

2.1.7 Instruction set simulator (ISS)

An ISS is a simulation model which mimics the behavior of a mainframe or microprocessor by "reading" instructions and maintaining internal variables which represent the processor's registers. Instruction simulation is employed for simulating the machine code of another hardware device to monitor and execute the machine code instructions, and to improve the speed of simulations involving a processor core (Patryk Zadarnowski, 2000).

2.2 Landmark Methodologies

2.2.1 Artemis (KNP-MOC)

Architectures and Methods for Embedded Media Systems (Artemis) is a simulation environment which is used for the modelling and simulation of multimedia systems (Pimentel et al., 2001). It supports two different approaches for simulation .i.e., System level Performance Analysis and Design Space Exploration (SPADE) and Simulation of embedded-system architectures for multilevel exploration (SESAME) (Terpstra et al. (2001)). Sesame is an approach for architectural exploration of heterogeneous embedded systems which is used when the design space is quite wide. Most optimal architecture candidates are picked at a higher abstraction level and then lower level details are added to go to the next lower abstraction level. This paves the way towards the final architecture called trajectory. Application and platform modelling and the mapping between them are shown in (Pimentel et al., 2001).

For application modelling KPN MOC (Pimentel et al., 2001) is used which nicely fits to the targeted media processing applications. KPN MOC does not allow the modelling of interrupts, thus the ability to model applications with time-dependent behaviour is limited.

Architecture models operate at the transaction level, and simulate the performance consequences of the computation and communication events generated by an application model. An architecture model is made from a library of generic building blocks which contain template performance models for processing cores, communication media, and various types of memory (Pimentel et al., 2001).

A designer shortlists the best candidate mappings for further evaluation. Thus, the optimal mapping of an application model onto an architecture model is the ultimate goal. An intermediate mapping layer is provided which consists of virtual processor components and FIFO buffers to communication with virtual processors. There is a one-to-one relationship between the Kahn processes in the application model and the virtual processors in the mapping layer. A mechanism is used to dispatch application events from a virtual processor to an architecture model component. This guarantees deadlock-free scheduling of the application events from different event traces (Pimentel et al., 2001).

Applications models are either generated by a framework called Compaan (Pimentel et al., 2001) or derived manually from sequential C/C++ code. An execution engine runs Kahn processes written in C++ as separate threads using Pthreads. Structure of application models is described in YML and architecture models are implemented either in Pearl or SystemC (Pimentel et al., 2001). For SystemC architecture models an add-on library to SystemC, called SCPEx is provided (Pimentel et al., 2001).

2.2.2 ARTS

Abstract system-level modelling and simulation framework (ARTS) (Mahadevan et al., 2005a, b) targets the domain of multimedia streaming applications and multiprocessor Systems on Chip (SOCs) for performance modelling and simulation. ARTS enable the system designers of heterogeneous multiprocessor SOCs to find the right partitioning of an application for a specific platform. The methodology aids the systems designers in analysing the following

- Performance of the network under various traffic and load conditions.
- The results due to mapping of various tasks to processors available from the platform.
- Usage of memory and other platform resources and effects of real-time OS (RTOS) selection.

The overall system model comprises of the application model mapped onto platform components. The platform is composed of multi-processor models, memories, communications and other platform resources while the application modelling employs state dataflow task graphs.

2.2.3 Baghdadi et al

This contribution (Baghdadi et al., 2002) presents a methodology for fast exploration of a large design space of computer systems. It has been implemented as an extension of hardware/software codesign flow for enabling fast exploration of multi-processor based systems from the early stages of the design. The approach is based on the codesign tool called MUSIC and system CODESIM simulator.

In the beginning, the system modelling is carried out. The system-level specifications are described in system-level description language (SDL). This results in heterogeneous multi-processor architectures comprising of both hardware and software. Afterwards, the SDL specifications are targeted by the designer for a particular architecture. In the last stage, the architecture-annotated SDL specifications are validated via simulation using CODESIM.

In the second stage, the partitioning and communication synthesis is performed. The best design alternatives, which meet the system constraints, are identified. This includes the mapping of system functions to hardware or software. During this phase, architectural choices are made which are further analysed to estimate the performance.

This leads to the third and final stage of prototyping. This step can be viewed as a combination of two steps. In the first step, a functional prototype is generated in VHDL/C which can be validated via co-simulation (Baghdadi et al., 2002). In the second step, the components of the architecture are targeted for generation of a cycle accurate model which can also be validated via co-simulation (Baghdadi et al., 2002).

2.2.4 Fornaciari et al

This methodology, proposed by Fornaciari et al. (2001), focuses on finding the best configuration of the memory hierarchy while avoiding the exhaustive analysis of the parameter space. The methodology is targets the domain of application-specific multimedia systems which usually have stringent energy and power constraints. While designing such systems, the prime focus is on exploring the design parameters of the memory subsystems. The methodology takes into consideration both energy and delay constraints by employing the metric called Energy Delay product (EDP). The methodology shows how the value of a given metric varies with the variation of the cache parameters. The methodology uses an iterative-local search algorithm for faster convergence to a near optimal design. The algorithm is based on the sensitivity analysis of the cost function with respect to the tuning parameters of the architecture of memory sub-system. During the sensitivity analysis, the optimisation of exploration parameters and

removal of less promising configurations from evaluation takes place. The optimization strategy of the algorithm is to start off from a suitable starting point within the design space followed by the sequential analysis of the parameters in order of decreasing sensitivity. All the parameters are not considered simultaneously for optimization. The algorithm focuses on the region of highest variation of the most important parameters dictated by the sensitivity analysis. The optimal solution is mostly reached due to the accurate sensitivity analysis is performed during the tuning phase.

2.2.5 Jabaer et al

Jaber et al. (2009) presented a methodology for investigating the impact of a shared system resource on a system's parameters for example delays, throughput and system resources utilization. The modelling is performed at an early stage before the realization of hardware and software architectures.

DIPLODOCUS framework is used for modelling both the application and architecture. This framework abstracts the application data via a concept called non-valued samples which results in very fast simulation. An open source tool called "TTool" which supports DIPLODOCUS is used by the methodology. DIPLODOCUS models both the application and architecture separately and using the Y-Chart approach (Jaber et al., 2009). A DIPLODOCUS application consists of a network of communicating tasks which can be connected via three communication semantics .i.e., the channels, events and requests. The channels exchange the abstract data samples; the events exchange signals while the requests trigger the execution of another task. The architecture consists of a network of physical resources which are abstracted by one of three types of architecture nodes .i.e., the computation nodes (for example CPUs, DSPs, and hardware accelerators etc.,), the communication nodes (for example busses, routers and switches etc.,) and the storage nodes (for example memories). The architecture resources can be instantiated via a library of abstract models. The models can be configured by setting the values of the performance parameters. The system model is formed by mapping the application model to the platform model. This is followed by the simulation of the system models using a SystemC based simulation environment. For each task, the simulation gives the best, worst and average execution time. The simulation also gives the utilization of all platform resources.

2.2.6 Koski

Koski (Kangas et al., 2006) targets not only the system-level modelling and exploration but also the implementation of multiprocessor SOCs based systems. The purpose of Koski is to create a real system which fulfils the design specifications. In other words, Koski produces an optimized physical architecture and mapping of the application to this architecture (Kangas et al., 2006). Koski design flow starts off by capturing the requirements of an application and architecture. This includes design constraints, for example the definition of cost function and the maximum allowed value. After the requirements specification, the functionality of the system is described in terms of an application model made in the UML design environment. This model is verified via functional simulations. The hardware architecture is also modelled in UML and is based on the application model of the targeted platform. The application and platform models are transformed to abstracted models via UML interface. This transformation results in faster simulation. Also the UML interface back-annotates the information of the optimized architecture to the UML design space for evaluation and refinement. This is followed by ar-

chitectural exploration consists of two phases. These phases are handled by the architectural exploration tools which examine the system models obtained from the UML-level. The exploration proceeds via analysis of the extensive set of architectures in which the used models are gradually refined. The portions of the UML description which were mapped to the processors in architectural exploration phase are passed to the automatic code generation. The generated low level software code and the platform component instances are combined for physical implementation which includes the integration of RTOS, generation of executable software and synthesis of hardware.

2.2.7 Lahiri et al

Lahiri et al. (2001b) presented a methodology for designing custom communication architectures for system on chip integrated circuits. The proposed technique is based on the hybrid trace based performance analysis approach. Firstly, an initial co-simulation of the system is carried out in which the communication is described at a higher abstraction level in terms of events and data transfers. This initial simulation gives an abstract set of traces which provide adequate information about the computations and communications of the components. Afterwards, the system designer specifies the communication architecture by: choosing a topology which consists of dedicated and shared communication channels (that are interconnected via bridges), mapping of the abstract communications to the paths in the communication architecture and customization of protocol which is used by each channel.

The traces extracted in the initial phase are presented in the form of Communication Analysis Graph (CAG). The CAG captures the computations, communications and synchronization information during the simulation of the overall system. A clustering algorithm is employed to select an initial mapping of the communication to a network topology, which is afterwards improved iteratively.

2.2.8 MESH

Modelling Environment for Software and Hardware (MESH) (Paul et al., 2003, 2005) is a performance simulator which allows a designer to efficiently explore the design space. The exploration of the design space takes place at the thread level instead of instruction level. The system performance is simulated by resolving the software execution into physical timing by using the high level models of processor capabilities in which the thread and message sequences are determined by the schedulers. The framework is based on a layered composition of threads, in which the dynamic logical threads are made on the top of physical threads. The physical threads model the hardware components of a platform and represent their computational power. The application software is modelled as logical threads. The executions of a dynamic number of logical threads are scheduled (by the scheduling layer of MESH) onto a processing element (for example a processor modelled as a physical thread). The logical threads are annotation with consume calls which represent the computational complexity of a software region. During the simulation, the threads are simulated for the amount of time determined by the thread annotations. Due to the contention for the shared resources, the delays are computed by grouping the requests to the same platform resources and sending them to the analytical model which returns the calculated delays. Since the large systems can have large number of small time slices, therefore, in order to increase the simulation speed, a minimum time slice can be defined which will of course compromise accuracy depending on its duration.

2.2.9 MILAN

Model-based Integrated Simulation (MILAN) is a methodology developed by (Mohanty et al. (2002)) for hierarchal design space exploration. The methodology consists of two phases. In the first phase, the design space is shrunk to a smaller design space via pruning techniques which first evaluate the preliminary design space and then limits it to a smaller number of designs based on the performance constraints and objectives. These pruning techniques are applied to highly abstract models of applications and platform components as well as performance constraints. In the second phase, the hierarchal simulation is performed by using a high level estimation tool and low level simulators. Low level simulators are used to perform component specific simulations for a given design. After gathering the component specific estimates, the high-level system wide estimator generates performance estimates for a complete heterogeneous embedded system. MILAN is implemented using Model Integrated Computing (MIC) which represents the system to be designed via domain-specific models (Mohanty and Prasanna , 2002)**.** MIC is in turn implemented using Generic modelling environment (GME), which is a meta-programmable toolkit for creating domain-specific modelling environments (Mohanty and Prasanna, 2002). The static semantics of a model are represented via Object Constraint Language (OCL) constraints. The design space is captured primarily via multi-aspect and hierarchal GME based Graphical models (Mohanty et al. (2002)). In order to explore and shrink the design space, a symbolic constraint specific methodology is applied.

2.2.10 Posadas et al

The methodology proposed by Posadas et al (2004) aims at the system level estimation of execution times from a system level performance description written in SystemC. It employs a C++ library and therefore does not require any change to the source code of the description. Also, the estimation is done by taking into account the characteristics of the system's platform which ensures high accuracy. Since the library is included within the usual simulation, the model of computation employed during the system design is preserved. The designer can target specific points in the code which are perceived important. The corresponding capture events are analysed later for verifying the timing constraints. The methodology separates communication from computation and thus the processes can interact with each other only via a set of pre-defined channels. The methodology works on the process segments which are conceived as task graphs rather than the basic blocks. When the architectural mapping takes place, each process segment is allocated to a hardware or software resource in the platform. The objects which contribute to the execution times of resources are redefined by overloading C operators and replacement of ordinary variable types with custom classes. A delay calculating function executes whenever a C++ object is executed. In case of hardware resources, the minimum and maximum execution times are estimated and also a weighted mean value is computed based on the input of designer. The overall system simulation proceeds via execution of segments and allowing the processes to sleep for an estimated time. The shortcomings of the approach include the requirement of the channel code to be modified via annotation functions and the constraint that only C++ applications can be simulated.

2.2.11 ReSP

Reflective simulation platform (ReSP) (Beltrame et al. , 2008) is a performance evaluation approach which focuses on multi-processor systems on chip. It is a component based methodology which works at a higher abstraction level and uses systemC and TLM. The compopnents used by ReSP are built on top of SystemC and the communication and hardwdecription are libraries(Beltrame et al. , 2008) of TLM. The methodology operates either interactively or automatically. In case of interactive operation, the syatem architecture is instantiated by the system designer via the commands sent by the interface while in the automatic mode, the architecture is described in an XML file from which the architecture model is created. This model can then be used for architectural exploration.

2.2.12 SPADE

SPADE (Lieverse et al., 2001a, and b) is an exploration strategy for signal processing architectures at the system level. It defines the platform architecture model by using trace driven execution units as platform architecture components.

SPADE models applications and architectures separately. Application models are functional which are relatively free of architectural aspects. Architecture models define the architecture resources such that they can be used for all applications from the benchmark set. This decoupling enables reuse of both application and architecture models and facilitates an explorative design process in which application models could be subsequently mapped onto architecture models. KPN MOC is used for application modelling. The execution of Kahn Process Network is deterministic which models the signal processing applications really well. It also facilitates an application programmer to combine communication primitives.

The architecture models don't model the functional behaviour but maintain functional correctness. The models can be constructed from a library of generic building blocks for different resources in architecture. A processing resource is built from two types of blocks. Trace driven execution unit (TDEU) and interfaces connect I/O ports of a TDEU to a communication resource (Lieverse et al., 2001a, b).

Each process is mapped to a TDEU. Mapping could be many-to-one. Trace entries of the processes are scheduled by the TDEU. Each process port is mapped onto an I/O port. Channels are mapped onto a combination of communication and memory resources and can also include user defined blocks (Lieverse et al., 2001a, b).

Simulation of the application model is based on the Pamela multi-threading environment. Simulation of the architecture model is currently based on TSS (Tool for System Simulation). Library of generic blocks consist of TSS modules. SPADE permits the use of user defined TSS modules and also provides an Application Programmers Interface (API) for application modelling (Lieverse et al., 2001a, b).

2.2.13 StepNP

StepNP focuses on network processors (Paulin et al. (2002)) and was developed by ST Microelectronics. The methodology provides a multiprocessor simulation model operating at a high abstraction

level alongwith a framework for network router application anda tooset for debugging and alalysing. The metholology does not employ any model of computation for the application models since it employs instruction set simulator for executing the application code. The methodology provides models for processors, NOCs and other components operating at the functional, transaction and cycle-accurate abstraction levels. Beltrame et al. (2007) developed a methodology basesd on StepNP for multi-accuracy power and performance modelling. The methodology enables StepNP to dynamic switch functionalities as well as communication and power consumption models operating at different abstraction levels.

2.2.14 TAPES

Trace Based Architectural Performance Evaluation with SystemC (TAPES) is a trace-based architecture exploration strategy which captures the functionality of architecture in the form of traces for platform resources. Wild et al. (2006) shows the high level design flow in which performance evaluation is a part of architecture exploration loop. Starting from a specification or a fully functional model and taking into account an initial architecture, a performance model is built that serves as input for performance evaluation step (Wild et al., 2006). Depending on analysis results, architecture is iteratively modified until design requirements are met. Architecture exploration loop results in the specification of an architecture which is used in the implementation phase. Functional validation is performed in parallel which guarantees compliance to specification (Wild et al., 2006).

KPM MOC is used for modeling Applications which model stream processing very well. It allows an application programmer to easily combine communication primitives. Write function is used to write data to a channel via a process port (Wild et al., 2006). An execute function only generates a trace entry, reporting on processing activities at the application level.

In platform model, the processing of tasks is replaced by their execution latencies on corresponding resources. Functionality of each resource is described as a sequence of processing delays interleaved with external transactions. Shared communication resources are not abstracted to same degree (Wild et al., 2006). To capture dynamics of resource conflicts due to transactions from parallel resources, mechanisms for contention resolution are implemented in communication resource models (Wild et al., 2006). Simulation model is instantiated at simulation start up from a library of abstract resource types. System architecture is built from modules which interact with transactions. For defining the architecture functionality, user has to provide functional specifications of modules used in the architecture. The simulation model captures the system functionality by specifying traces for all architecture resources (Wild et al., 2006).

As per portioning decision, each resource consists of one or many traces which are related to different processing sequences. Trace driven interaction of architecture resources enables the simulation of system behavior. Models of shared resources implement mechanisms for resolution of conflicting accesses which results in stretched traces due to arbitration latencies and processing time in called modules. In a write transaction, the required data is first written to the accelerator which is directly followed by the read of the result. After the write operation, the execution of corresponding accelerator trace is triggered. The subsequent read transaction blocks the CPU until accelerator has finished and the result is transferred back to the CPU (Wild et al., 2006). KPM MOC partitions an application into a set of parallel communicating processes. SPADE also provides an API for application modeling.

2.3 Overview of ABSOLUT

ABSOLUT uses the Y-chart approach (Kienhuis, 1997) for SLPE and consists of the application workload model and execution platform as shown in UML diagram of Figure 7. The figure follows the UML notation for showing the containment relationship between the Performance model and Application and execution platform model.

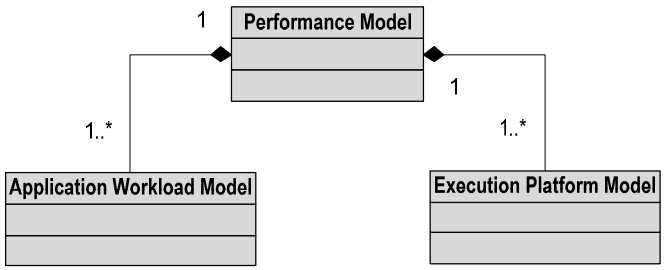

Figure 7: Main parts of an ABSOLUT performance model

The complete performance model is formed by mapping the application workload models to the execution platform model which is simulated to obtain the performance numbers which are analysed by the system designer. If the results do not meet the design constraints, the platform models or application models or both are changed in the next iteration as elaborated in Figure 4.

2.3.1 Structure of Application Workload Models

The workloads models consist of three layers .i.e., main workload, application workload and function workload as shown in UML disgram of Figure 8. The figure follows the UML notation for showing the containment relationship between different layers of Application workload models.

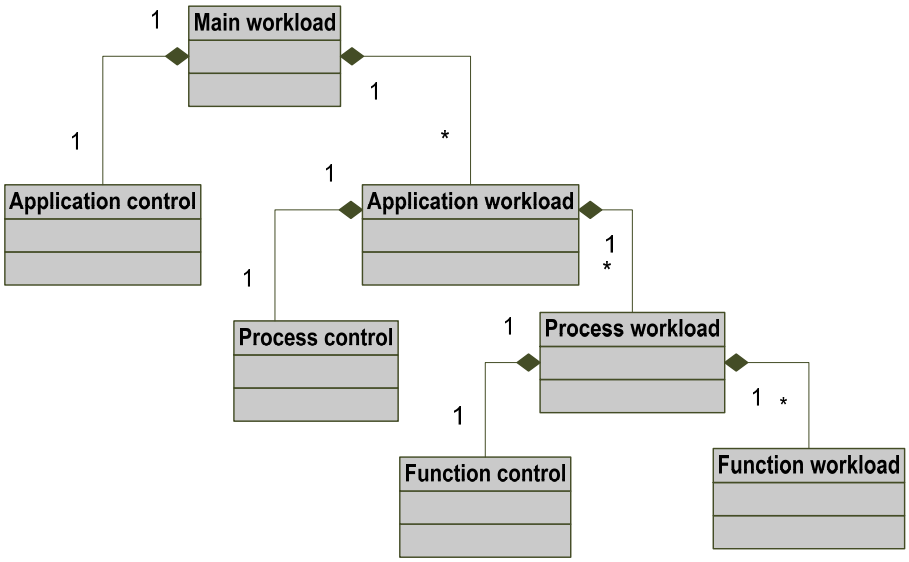

Figure 8: The ABSOLUT application workload model layers

The topmost layer consists of the main workload which is composed of one or more application workloads, each of which corresponds to an application supported by the system:

$W= \{C_a, A_1, A_2 \ldots A_n\}$ (1)

Where A_1, $A_2 \ldots A_n$ represent different application workload models and Ca is the control. In the second layer, each application workload is refined to one or more (platform-level) service or process workload models. Each of these (service or process) workloads are denoted by P_i:

$A_i= \{C_p, P_1, P_2 \ldots P_n\}$ (2)

Where C_p is the control and P_1, $P_2 \ldots P_n$ show service or process workload models. In the third layer each Process or service workload is represented as a composition of one or more function workloads:

$P_i=\{C_f,F_1,F_2 \ldots F_n\}$ (3)

where C_f is the control between function workload models .i.e., $F_1,F_2 \ldots F_n$. The ABSOLUT OS model of the platform handles the scheduling of workload at the process level. The function workloads are control flow graphs:

$F_i= (V, G)$ (4)

where the nodes $v_i \in V$ represent the basic blocks and $g_i \in G$ represent the branches. Each basic block is an ordered set of load primitives used for load characterization.

2.3.2 Application workload modelling techniques

ABSOLUT application workload models can be generated via three methods, .i.e., analytical, measurement based, and trace based and compiler based workload generation. The analytical workload

generation requires moderate modelling effort and is based on the analysis or functional description of the algorithms. After analysis, the number of operations required to perform the application tasks are estimated. These operations are used to estimate the number of abstract instructions in the corresponding ABSOLUT workload models (Kreku et al., 2008b). The measurement based workload modelling technique is based on the extraction of data from the partial traces of the modelled use-cases (Kreku et al., 2008b). The trace-based application workload generation method generates workloads by tracking the instructions executed by the processors while running the modelled application (Kreku et al., 2008b).

The compiler-based workload generation method uses a tool called ABStract INstruction exTraction Helper (ABSINTH) (Kreku et al., 2008b). It is based on GNU Compiler Collection (GCC) version 4.5.1 with additional passes in the compiler middle end which enable workload model generation. The workload generation takes place in three distinct phases. In the first phase, application source code is compiled via ABSINTH with profiling information extraction enabled. This data is used by ABSINTH in the second phase for statistical modelling of branches probabilities and extraction of number of loop iterations. In the second phase, the selected use case is executed by running the application in order to produce the profiling data. In the last phase, the source code is recompiled to produce workload models which are based on actual true execution of the application (Kreku et al., 2008b). ABSINTH generates one workload model for each function in the application source code. These models can contain calls to other function workloads without the knowledge of implementations of these workloads. Before compiling the models for simulation, they are post-processed with ABSINTH manager. It is a Python script which detects function dependencies from a set of workload functions and modifies the files by linking them in the order in which the functions were called in the application.

2.3.3 Platform Modelling

The platform model is also layered and consists of three layers, .i.e., component layer, subsystem layer and the platform architecture layer as shown in Figure 9. As stated before, the figure follows the UML notation for showing the relationship between different entities.

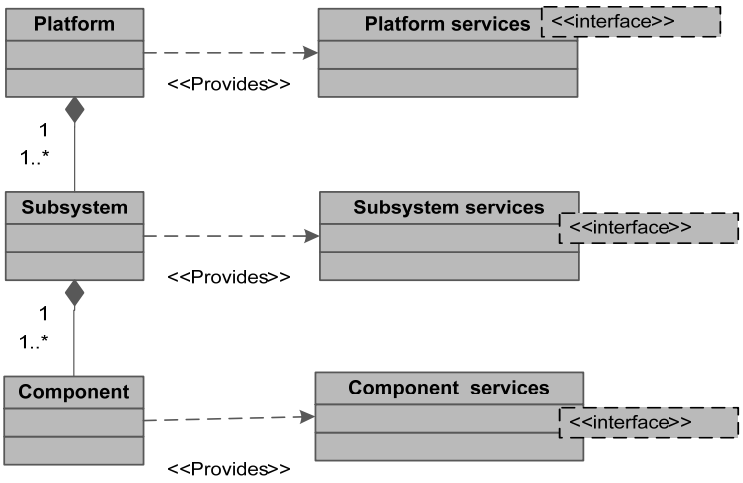

Figure 9: The ABSOLUT platform architecture model layers.

The component layer is composed of processing, storage and interconnection elements. The subsystem layer is built on top of the component layer. This layer shows the components of the system and the way they are connected. The platform architecture layer which is built on top of the subsystem layer incorporates platform software and also serves as portal which links the workload models to platform during mapping process (Kreku et al., 2008b).

2.3.4 Mapping and co-simulation

The workload models are mapped to the execution platform model which involves the selection of the part of platform which will execute a particular workload model. This is performed during the initialization of a workload model by passing a pointer to correct host in each workload constructor. Mapping is done at each layer .i.e., by mapping application workloads to subsystems, process workloads to OSs inside subsystems and functional workloads to processing units. The system model is built on Linux platform via CMake (Kreku et al., 2008b) and the Open SystemC Initiative (OSCI) SystemC library (Kreku et al., 2008b). The simulator is executed from the command line. During simulation, the progress information is printed to the standard output and after the completion of simulation; the gathered performance results are displayed.

2.4 Comparison of Methods and Tools

Table 1 presents the way salient SLPE approaches can be compared on the basis of different features. Four different aspects of the SLPE approaches are considered to evaluate their feasibility for extension to the domain of distributed networked embedded systems and applications. These aspects include the Modelling Style, Languages and frameworks used feasibility for validation of non-functional properties of distributed applications and the targeted domain.

2.4.1 Modelling Style

Different design space exploration methodologies employ either static (analytical) or dynamic (simulation) estimation methods. Usually, the static estimation methods shrink the vast design space briskly but the models employed are very coarse. The models used by dynamic estimation methods are more accurate and detailed but are slow at pruning the vast design space. In other words, the static estimation favour speed instead of accuracy for design space exploration while the dynamic exploration methods favour accuracy instead of speed. Some methods utilize a combination of static and dynamic simulation for exploiting the advantages of both methods. Static methods employ analytical or highly abstract models of applications and usually ignore the dynamic behaviour of the application which depends on the input data. As a result, the static methods don't offer the level of accuracy for exploration and communication scheduling as the dynamic simulation methods. Most of the salient performance simulation approaches utilize the dynamic estimation approach while ARTEMIS, MESH and KOSKI use both static and dynamic estimation methods as shown in Table 1. It was observed that different methodologies model the applications and platforms at various levels of abstraction and refinement. ABSOLUT employs layered application and platform models. The platform models operate at the transaction level while the lowest layer of application models comprise of abstract instructions. Detailed description of ABSOLUT modelling methodology is provided in (Kreku et al., 2008b).

Some methodologies for example SPADE; TAPES, ARTEMIS and KOSKI model applications as Kahn Process Network (KPN) Model of Computation (MOC). Platform models in SPADE are instantiated via a library of generic building blocks which model different resources in the platform. The processing elements in the platform are modelled as TDEUs. Each process of the modelled application is mapped to a TDEU in the platform (Lieverse et al., 2001a, b). TAPES abstracts the processing of tasks by their execution latencies on the corresponding resources in the platform (Wild et al., 2006). Further details of platform modelling in TAPES are mentioned in (Wild et al., 2006).

The architecture models in ARTEMIS operate at transaction level, which simulate the computation and communication events that are generated by the application model (Pimentel et al., 2001). An architecture model is made from a library of generic building blocks which contains templates of performance models for different platform elements (Pimentel et al., 2001). KOSKI employs UML for modelling platform which is later on transformed to an abstract model via UML interfaces (Kangas et al. , 2006).

ARTS employs static data flow graphs (SDFG) MOC to model the applications while the architecture models operate at transaction level and simulate the performance consequences of the computation and communication events generated by the platform (Mahadevan et al., 2005a, b).

Baghdadi et al. (2000) describes the system-level specifications in SDL which results in heterogeneous multi-processor architectures consisting of both hardware and software components. SDL can model a variety of embedded software applications (both real time and non-real time).

Fornaciari et al., (2001,2002) uses software execution profiler for the cycle accurate simulation of the application while the data and address bus streams are generated via a dynamic tracer.

Jabber et al., (2009) model applications via DIPLODOCUS tool. A DIPLODOCUS application comprises of a network of tasks which communicate via communication semantics defined by the meth-

odology. The architecture comprises of a network of physical resources which are abstracted by one of three types of architecture nodes, .i.e., the computation nodes (for example CPUs, DSPs, and hardware accelerators etc.), the communication nodes (for example busses, routers and switches etc.) and the storage nodes (for example memories) (Jabber et al., 2009) .

Some performance evaluation methodologies such as ReSP (Beltrame et al. (2008)) uses ISS for processors and therefore there are no models for applications. The modelling styles of landmark performance simulation methodologies are listed in Table1.

2.4.2 Languages, Standards and Frameworks

The landmark performance evaluation methodologies described in Table 1 use a variety of widely used modelling, scripting and Programming languages such as C, C++, PEARL, UML and XML for various modelling purposes.

Some methodologies use specification languages such as SDL or LOTOS OSI specification language for system-level specifications. Some methodologies such as MILAN use other tools such as HiPerE and DESERT for modelling and simulation purposes. Lahiri et al. (2001 a, b) uses POLIS and PTOLEMY frameworks for designing communication architectures of SOCs. It was observed that some methodologies use modelling languages and simulation frameworks such as SystemC which is widely used for the system-level modelling, architectural exploration, performance modelling etc., of electronic systems. The programing and modelling languages used by the landmark methodologies are shown in Table 1.

2.4.3 Non-functional properties validation

The distributed embedded systems support applications which consist of many components running on different networked devices. In such cases, the application components communicate via transport, data link and (possibly) middleware technologies. These distributed applications are generally message based or streaming applications which satisfy the end-user requests by (in turn) requesting one or more services provided by different devices which implement these services. Therefore, the end-user experience is not merely a consequence of the application implementation since the transport protocols, data link protocols as well as physical layer plays a key role in the end-user experience since the end-to-end delays and packet/frame errors at these layers can deteriorate the end-user experience. Therefore, for a methodology to be able to estimate reliable performance numbers for distributed applications, it must model these OSI model layers with sufficient level of detail. These models must preserve the functionality to a level that the estimated delays show a close correlation with delays estimated by network simulators such as OMNeT++ and ns-2.

It has been noticed that some methodologies are totally focused on one particular domain of applications and systems. The methodologies such as ARTEMIS, KOSKI, SPADE and TAPES which model applications via KPN MOC are limited to the performance estimation of streaming applications since KPN models only model streaming applications very well. Therefore a wide variety of message based distributed applications cannot be modelled via these methodologies. Some methodologies such as Lahiri et al. and ARTS use other models of computations such as CAG and SDFG for modelling

applications. It was also observed that some methodologies use their own model of computation for describing applications while the others such as ABSOLUT employ a layered application model (Kreku et al., 2008b).

In all the methodologies which employ a model of computation to describe the applications, we observe that the functionality of the transport and data-link layer has been abstracted by the communication paradigm employed by the MOC. This means that the non-functional properties of a distributed application (such as end-to-end delays and packet/frame loss rate) cannot be reliably estimated since the functionality of transport and data-link layers have been swapped by that of the communication means defined by the employed MOC. All the MOCs use simple channels for communication among processes for example KPN MOC use simple FIFO channels for passing synchronization tokens among processes. On the other hand in networked devices, data-link layer MAC protocols resolve the contentions for occupancy of the common channel (wired or wireless). The level of abstraction used to model channels should be comparable to the abstraction level employed by network simulators such as OMNeT++ and ns-2 (Khan et al., 2011b). The MOCs, targeted application domain and the availability of models for transport, data-link and Middleware technologies models are highlighted in Table 1 for the landmark performance simulation methodologies.

2.4.4 Targeted systems Domain

Some methodologies are totally focused at the performance evaluation of a particular domain of computer systems, for example TAPES and Fornaciari et al. target single processor based systems, SPADE and Baghdadi et al. only target Multi-Processor based systems while Lahiri et al. is only focused on the performance evaluation of on-chip communication architectures. StepNP (Paulin et al. (2002)) and ResP (Beltrame et al. (2008)) focus on multi-processor based SoCs. ReSP is a hardware simulation platform which is mainly aimed at the architectural exploration of Multi-Processor Systems-On Chip systems while StepNP concentrates on Network processors.

The other landmark methodologies are also focused on only two of the three performance simulation objectives, .i.e., and performance evaluation of single processor based SoC, multi-processor based SoC and on-chip communication architectures. It means that they cannot be used for the performance evaluation of distributed embedded systems. It was observed that only three methodologies .i.e.; ABSOLUT, ARTEMIS and KOSKI cover all these three domains.

The use of multiprocessor based platforms is increasing in high-end mobile handheld devices such as smart phones and internet tablets while on the other hand in case of wireless sensor networks very low power and single processor based systems are used. Hence, for the performance simulation of a wide variety of distributed embedded systems, it is important that the methodology is not restricted to certain type of platforms (single or multiprocessor based) or a particular aspect of a platform such as performance evaluation of on-chip/intra-platform communication architectures. The targeted domains (of embedded systems) of landmark performance evaluation methodologies are listed in Table 1.

Table 1: Analysis of salient design space exploration methodologies

Name of approach	Modelling Styles, Tool Coupling and Performance Estimation					Languages, Standards and Other Frameworks Used		Non-Functional Properties validation				Targeted System Domain			
												Non-Distributed Systems			
	Performance estimation type	Application Model	Architecture Model	Tool Coupling	Simulation Based Approaches	Programming/Modelling Languages and Standards Spanned	Other Approaches/Frameworks Used	Model of computation	Targeted Application Domain	Transport and Data link Models	Middleware Layer Workload Models	Single Processor based systems	Multi-Processor based SOC	On-Chip/Intra-Platform Communication Architectures	Distributed Networked Systems
ABSOLUT	D	X^{11}	X^{12}	X^{13}	*X*	C/C++/ SystemC 2.2/TLM2.0/ UML		TLM	*A*			X	X		
ARTEMIS	D/S	X^{21}	X^{22}	X^{23}	X	PEARL, SystemC, RTL	*SPADE, SESAME*	KPN	*ST*			X	X		
ARTS	D	X^{31}	X^{32}		X	SRTS Scripting Language, SystemC		SDFG	*ST*				X		
Baghdadi et al	D	X^{41}	X^{41}		X	C,RTL, SDL,	*MUSIC, CODESIM*	N	X^{42}				X		

Fornaciari et al	D	*X*[51]	*X*[52]		X	C/C++	MEX, Shade, X[53]	N	ST			*X*[54]			
Jaber et al	D	*X*[61]	*X*[62]		X	*UML, SystemC, LOTOS OSI Specification Language*	*DIPLODOUS*	N	X[63]			X	X		
Koski	D/S			*X*[71]	X	TUT UML profile,XML	Existing Code Generators and Compilers	KPN	A			X	X		
Lahiri et al	D		*X*		X	Languages/ Tools used by POLIS & PTOLEMY	POLIS, PTOLEMY	CAG	X[81]						
MESH	D/S		*X*		X	C		X[91]	A			X	X		
MILAN	D	X[101]	*X*		X	Languages/ Tools used by DESERT & HiPerE	DESER, HiPerE, X[102]	X[103]	A						
Posadas et al	D	*X*[111]			X	C++/SystemC		X[112]	*X1*[113]			X	X		
ReSP	D	*X*[121]	*X*[122]		X	C++/SystemC/IDL/Python/XML	GCCXML/ArchC	TLM	*X*[123]				X		
SPADE	D	*X*	*X*		X	C/C++	YAPI,TSS	KPN	*ST*				X		

StepNP	D	*X*[131]	*X*[132]		X	Sys-temC	OCP Protocol	TLM	*X*[133]				X		
TAPES	D	*X*	*X*		X	Sytem C,XML		KPN	*ST*			X			

Abbreviations Used In the Table

A — No restriction as per our assessment. Also no mention of a particular application domain by the authors.

ST — Streaming Applications.

D — Dynamic

S — Static

N — No MOC such as KPN, CAG, TLM and CDFG used or affectively adapted by the methodology. The modelling of applications is elaborated in the corresponding reference in the second column.

TML — Transaction Level Modelling

Methodology Specific Information Used in Table

ABSOLUT

X[11] ABSOLUT uses layered application workload models consisting of application, process and function workload layers. The function workloads consist of abstract instructions and control.

X[12] The platform model is also layered and consists of three layers, .i.e., component layer, subsystem layer and the platform architecture layer.

X[13] The ABSINTH-2 tool of ABSOLUT uses Valgrind for the workload generation of external libraries.

ARTEMIS

X[21] Applications are modelled as KPNs which are either generated by a framework called Compaan or derived manually from sequential C/C++ code (Pimental et al., 2001).

X[22] Architecture models operate at the transaction level and simulate the performance consequences of the computation and communication events generated by an application model. An architecture model is made from a library of generic building blocks containing template performance models for processing cores, communication media, and various types of memory.

X^{23} ARTEMIS uses Laura tool set for automatic generation of VHDL code from application models.

ARTS

X^{31} Applications are modelled using static dataflow task graphs.

X^{32} The platform consists of multi-processor models, memories, communications and other platform resources.

Baghdadi et al

X^{41} Information related to application and architecture modelling and tool coupling is provided in Chapter 2.2.3.

X^{42} The system-level specifications are described in SDL. This results in heterogeneous multi-processor architectures comprising of both hardware and software. SDL does not explicitly specify any particular domain or restriction as far as its ability to model software is concerned.

Fornaciari et al

X^{51} The simulation framework is based on a software execution profiler for cycle-accurate instruction set simulation of the application and a dynamic tracer for generation of data and address bus streams.

X^{52} Design space exploration is focused on the processor to memory communication through the memory hierarchy and includes configurable bus and memory models, with the latter having behavioural models of on- and off-chip level 1 and 2 caches and main memory. The bus and memory models use the bus traces from the software execution profiler as input.

X^{53} The architecture is explored by using a tool called MEX which simulates the execution of a program compiled for the Sparc V8 architecture within configurable memory architecture. MEX exploits the Shade (Fornaciari et al., 2001) library to trace the memory accesses made by a SPARC V8 program and consequently simulates the target memory architecture to obtain accurate memory access statistics. The MEX tool uses a C++ based tool called Shade as described in Fornaciari et al. (2001).

X^{54} Design space exploration is focused on the processor to memory communication through the memory hierarchy. The technique aims at finding the best platform configuration for the application without an exhaustive search of the parameter space. The parameters for the exploration include cache size; block size and instruction cache associativity.

Jaber et al

X^{61} Applications are modelled via DIPLODOCUS. A DIPLODOCUS application consists of a network of communicating tasks which can be connected via three communication semantics .i.e., the channels, events and requests. The channels exchange the abstract

data samples; the events exchange signals while the requests trigger the execution of another task (Jaber et al, 2009).

X^{62} The architecture is modelled as a network of physical resources, including computation, communication and storage nodes. All resources have parameters like processing capacity in millions of cycles per second or memory size in bytes (Jaber et al, 2009).

X^{63} Not restricted to any particular application domain as per our assessment. Applications are modelled by using the data and functional abstraction mentioned by the authors in (Jaber et al, 2009).

KOSKI

X^{71} KOSKI uses existing compilers and code generators for refining the applications to the final processing elements.

Lahiri et al

X^{81} This methodology is only targeted at the design of custom communication architectures for system on chip integrated circuits.

MESH

X^{91} The framework is based on a layered composition of threads, with the dynamic logical threads made on the top of physical threads. The physical threads model the hardware components of a platform and represent their computational power. The application software is modelled as logical threads. The execution of a dynamic number of logical threads is scheduled (by the scheduling layer of MESH) onto a processing element (for example a processor which is modelled as a physical thread).

MILAN

X^{101} Applications are modelled as trace files by Hyper which consist of a list of communication and computation tasks (Posadas et al., 2004).

X^{102} DESERT and HiPerE are used for rapid design space exploration. DESERT shrinks the design space by shortlisting designs and HiPerE estimates the performance.

X^{103} The methodology proposed by Posadas et al (2004) is only aimed at estimation of execution times from a system level performance description written in SystemC. No MOC is employed by this methodology.

Posadas et al

X^{111} This methodology only aims at system the estimation of execution time from a system level performance description written in SystemC and therefore does not employ application models. It estimates the execution time of the application via a C++ library.

X^{112} Applications are modelled as a set of Processes which can only interact with each other

via predefined channels.

X[113] Only C++ applications can be simulated.

ReSP

X[121] ReSP uses ISS for processors and therefore there are no models for applications.

X[122] The methodology provides a library of platform component models such as NOCs, busses, memories as wll as functional and cycle accurate processor models.

X[123] ReSP is a hardware simulation platform which is mainly aimed at the architectural exploration of Multi-Processor Systems-On Chip systems.

StepNP

X[131] Beltrame et. Al. proposed a methodology for mapping applications on platforms based on StepNP which facilitates a system designer for co-explortion of the design space. Firstly, the application source code written in C-programming language is natively profiled in a workstation. Afterwards, the application and platform model are simulated via StepNP.

X[132] The StepNP environmenst provides highly abstract multiprocessor architecture simulation models, network routers and a set of tools for debugging and analysis.

X[133] Concentrates mainly on Network processors.

2.5 Summary

In this Chapter, the important aspects of salient SLPE approaches were elaborated. We observed that these methodologies employ a variety of tools and modelling languages and mostly focus on a few modelling (targeted system domain) objectives shown in Table 1. Different methodologies describe the application and platform models at different levels of abstraction and employ different models of computation for describing the application models. Also, some of the methodologies use third party tools for modelling or simulation purposes and some provide tools coupling for extending the usability of the methodology for other simulation objectives. In the next Chapter, we further investigate the feasibility of landmark performance evaluation approaches described in this Chapter for the SLPE of distributed embedded systems.

3. Towards Performance Evaluation of distributed systems

3.1 Requirements for SLPE of distributed systems

After investigating the protocols and technologies employed by the distributed embedded systems in Chapter 1 and salient SLPE approaches in Chapter 2, we conclude that, in order to validate the NFPs of distributed embedded systems at an early stage, a SLPE methodology must provide a number of models/features. Also, in order to span different domains of distributed embedded systems, a SLPE methodology must take into account all the dimensions shown in Figure 1. We therefore conclude from the literature survey that the following protocol models, features and tools must be provided by a SLPE methodology in order to perform SLPE of distributed embedded systems in different domains.

I. ***MOC Agnostic:*** The methodology should not employ a specific MOC for modelling applications and platforms. Using a MOC restricts the methodology to a particular domain of systems, for example the methodologies which use KPN MOC for application modelling are usually targeted only at streaming applications (Lieverse, 2001a, and b).

II. ***Multithreaded Application Modelling:*** For performance evaluation of multi-threaded applications, the methodology must model the multi-threading support (Saastamoinen , 2011a).

III. ***Physical Layer Models:*** Physical layer models such as channel models, coding and modulation models to evaluate the contribution of these layers in non-functional properties of distributed applications (Khan et al., 2011b).

IV. ***Transport Layer Models:*** Functional models of OSI Data-link and Transport layer protocols for evaluating their contribution in non-functional properties such as end-to-end packet and frame delays (Khan et al., 2011b).

V. ***Performance Evaluation of Protocols:*** The methodology must be capable of evaluating the performance of protocols operating on a particular layer of the OSI model in isolation just like widely used network simulators for example OMNeT++ and ns-2 by abstracting application workload models via using traffic generators (Khan et al., 2011b).

VI. ***No domain Restriction:*** In order to span the domain of distributed systems such as WSNs, the methodology must be capable of evaluating the percentage utilization of platform by Data-link and Transport Protocols. WSNs in particular employ highly efficient and specialized Data-link protocols to reduce power consumption as described in Chapter 1.

VII. ***Easy Modelling of New Protocols:*** Modelling of new Data-link and Transport layer protocols should only focus on the functional aspects of the protocols and the designer must not waste time in modelling the features that are similar to the existing protocol models (Khan et al., 2011b, c, and d).

VIII. ***Workload Model Generation of User-Space code, External Libraries and System Calls:*** Since, from an implementation perspective, all the applications processes use user-space code, external libraries, background processes and system calls, therefore, the methodology must provide tools and methods for generating the workload models of not only the user space code but also the external libraries, background processes and system calls

(Kreku et al. , 2009, 2007, 2008 (a, b)), (Saastamoinen, 2011a) (Khan et al. , 2012a).

IX. ***Workload Generation of middleware technologies:*** It must be capable of workload extraction of API functions of the various Middleware technologies such as NoTA SOA. This will enable the methodology to span the domain of distributed streaming and context aware applications (Khan et al., 2011c).

X. ***Detailed as well as Highly Abstract Workload Modelling:*** The methodology must provide/define application workload modelling tools/techniques for generating the application workload models with varying degrees of refinement and detail. The more refined and detailed workload models result in slower simulation speed due to increased structure and control while the less detailed workload models usually result in faster simulation speed (Kreku et al. , 2009, 2007, 2008 (a,b),2004 (a,b)) at the expense of accuracy. Once this is achieved, the system designer can freely choose the workload models that will result in more accurate or faster simulation.

XI. ***Integration of Application Design and Performance Evaluation:*** For early phase evaluation of the distributed applications, the methodology must automate the workload extraction process by seamless integration of application design and performance simulation phase. This can be achieved if the application and workload modelling phases are linked in such a way that application models act as blue print or starting point for the application workload models. The proposed technique must be experimented with modern SOAs such as GENESYS and NoTA (Khan et al., 2011(c, d)) (Khan et al., 2009).

XII. ***Reporting of Results for Different Protocol layers:*** The results reporting mechanism must elaborate the contribution of protocols operating at different layers to the non-functional properties separately, for example the processing delays, end-to-end delays and percentage utilization of the platform must be reported for each layer of the OSI model as well as middleware technologies (Khan et al., 2011 (b, c, d)).

XIII. ***Easy Modelling of Probes:*** New performance measuring probes should be easy to model and integrate into the framework so that system designer can easily evaluate the performance of specific protocols, middleware technologies or platform components in the distributed embedded system.

XIV. ***Non-Functional Properties Validation:*** The non-functional properties of the system must be carried through the application design phase and validated by the SLPE approach. The non-functional properties are usually modelled and elaborated in the application model views (Khan et al., 2009, 2011 (b, c, d)).

XV. ***Model of Computation Instantiation:*** While modelling applications belonging to domains where the Transport and Data-link technologies can be abstracted out, the methodology must be capable of instantiating model of computations for application modelling for faster simulation speed (Khan et al. , 2011a). For example streaming applications can be modelled well via KPN MOC if the performance simulation objective is other than validation of non-functional properties which are not affected by end-to-end packet delays or frame loss rate etc., which cannot be estimated with reasonable accuracy without Transport and MAC models.

3.2 Feasibility of Existing SLPE Approaches

As shown in Table 1, none of the methodologies considered in the survey is capable of providing highly accurate estimates of the non-functional properties of distributed applications. The reason is that in case of distributed embedded systems, the transport, data-link and middleware technologies (possibly) contribute to the non-functional properties such as end-to-end frame and packet delays. In order for a methodology to accurately estimate the effects of these protocols, it must employ functional MAC and Transport Protocols. As shown in Table 1, majority of the salient SLPE methodologies are limited to a particular domain or aspect of modelling of embedded systems. Due to these limitations, they cannot be considered for the performance evaluation of distributed embedded systems in different domains.

Only three out of all the approaches mentioned in Chapter 2.2, .i.e., ABSOLUT, ARTEMIS and KOSKI are not restricted to any particular domain of embedded systems. Furthermore, out of these approaches, ARTEMIS and KOSKI use KPN MOC for modelling applications which can only model streaming applications

well (Lieverse, 2001a, b) (Zivkovic, 2002). Also, KPN MOC does not allow the modelling of interrupts and thus the ability to model applications with time-dependent behaviour is limited. Therefore, out of all the system level performance evaluation methodologies, only ABSOLUT is not limited to any particular domain of embedded systems or applications as shown in Figure 10.

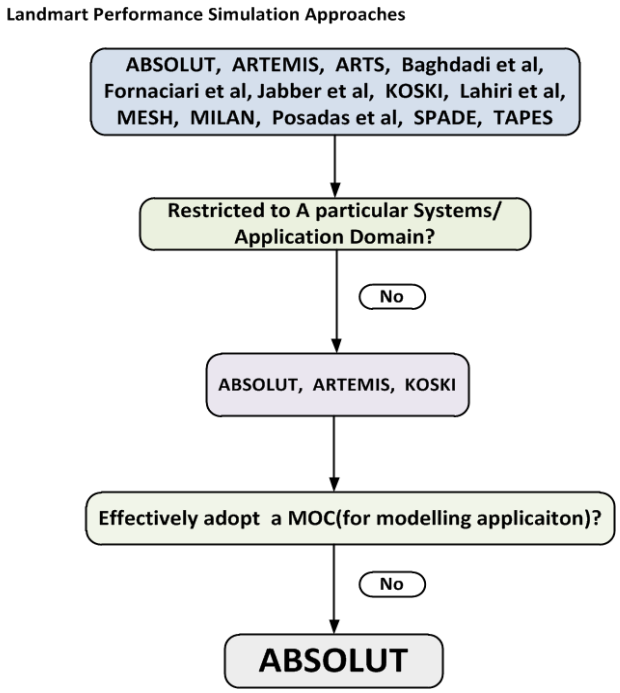

Figure 10: Selecting a SLPE approach for SLPE of distributed systems

In the next chapter, we elaborate the feasibility of ABSOLUT for SLPE of different domains of distributed embedded systems. This is achieved by identifying the requirements mentioned in Chapter 3.1 which are already provided by ABSOLUT. The features mentioned in Chapter 3.1 which are not provided by ABSOLUT have been modelled and integrated to ABSOLUT during the research presented in this thesis. After integration of the tools and models developed during the research, ABSLOUT can be used for the SLPE of distributed embedded systems in different domains.

3.3 Enhancements needed for ABSOLUT

ABSOLUT uses TLM 2.0 and SystemC for modelling platform components only, it is not restricted to a particular domain or TLM 2.0 for modelling applications. Also, ABSOLUT provides an OS model which is hosted on one or more processor models in the platform. The OS model consists of a scheduler and provides the possibility to model different OS Services. The OS scheduler schedules the application processes and a variety of services can be implemented by the system designer by implementing the Generic_Service Interface. The scheduling of the implemented services closely mimics the way services are scheduled by the widely used platforms (mostly via scheduling queues). The challenges in designing and implementing these services in SystemC (as ABSOLUT OS Services) requires a thorough knowledge of the Transport, Data-link and Physical layer technologies and event driven simulation paradigm.

Also, ABSOLUT workload generation tools provide a high level of automation for the extraction of application workload models in order to test their feasibility on a variety of platforms. This is especially useful when the source code of the application is available. In the absence of source code, the ABSOLUT workload models can be created via an analysis of the algorithmic details and the control of the application. For fast performance evaluation and architectural exploration, the ABSOLUT workload modelling phase and performance simulation phase can be integrated. This reduces the time and effort involved in the performance simulation. Also, the non-functional properties must be carried through the application design phase and validated by the performance evaluation phase. This seamless integration of application design and performance simulation phase has been demonstrated for different SOA based application design methodologies such as GENESYS and NoTA.

In ABSOLUT, the cycle accurate platform component models are not avoided and cycle approximate models used for faster simulation speed and brisk iterations in the architectural exploration phase (Kreku et al., 2008b). Before the start of the research, ABSOLUT has been successfully employed for the performance simulation of NoC based SOCs but it has never been employed for the SLPE of distributed embedded systems involving multiple devices. We now list the features mentioned in Chapter 3.1 that were provided by ABSOLUT beforehand before the start of the research presented in the thesis. We also provide the references to the research articles which describe these contributions. This information is shown in Table 2 and Table 3.

Table 2: Features Provided by ABSOLUT beforehand for the performance evaluation of distributed embedded systems.

Feature		***ABSOLUT***	
		Description	**References**
I-	MOC Agnostic	***X***	(Kreku et al. 2008b)
II-	Multithreaded Applications Modelling	***N***	
III-	Physical Layer Models	***N***	
IV-	Transport Layer Models	***N***	
V-	Performance Evaluation of Protocols	***N***	
VI-	No domain Restriction	***X[1]***	(Kreku et al. 2008b)
VII-	Easy Modelling of New Protocols	***N***	

VIII- Workload Model Generation of User-Space code, External Libraries and System Calls	X^2	(Kreku et al. 2008b), (Saastamoinen, 2011b)
IX- Workload Generation of middleware technologies	X^3	(Kreku et al. 2008b), (Saastamoinen, 2011b)
X- Detailed and Highly Abstract Workload Modelling	X^4	(Kreku et al. , 2009, 2007, 2008 (a,b),2004 (a,b))
XI- Integration of Application Design And Performance Evaluation	N	
XII- Non-Functional Properties Validation	X^5	(Kreku et al. 2008b)
XIII- Reporting Of Results for Different Protocol layers	N	
XIV- Easy Modelling of Probes	N	
XV- Model of Computation Instantiation	N	

Table 3: Description of the terms shown in Table 2.

X	Feature completely provided
N	Feature, related model or tool not provided
X^1	Restricted to the domain of non-distributed systems (single device based systems). But not restricted to any Application Domain.
X^2	Cannot generate the workload models of System calls for example Berkeley Software Distributions (BSD) API functions etc.
X^3	Only if the middleware is implemented as an external library. It cannot generate the workload of middleware technologies if it is implemented as OS services or system calls or runs as a background process for example NoTA operates in Daemon mode.
X^4	Only for user space code and external libraries. It cannot generate the workload for system calls.
X^5	Only for non-distributed applications running on a single device. In other words cannot validate the non-functional properties due to MAC and Transport protocols in use-cases which involve two or more devices.

From Table 2 it is clear that a lot of features mentioned in Chapter 3.1 are either completely or partially absent in ABSOLUT. The missing features have N in the corresponding "Description" column field in Table 2.We now mention the thesis chapters which describe the modelling and integration of these missing features. The modelling and integration of Feature II .i.e., Multithreading Support is described in Chapter 4. The modelling and integration of Features III → VII are covered in Chapter 5. This chapter describes the way MAC and transport protocols etc., are modelled and integrated to ABSOLUT. It also describes the way new MAC and Transport protocols can be easily modelled and integrated to ABSOLUT by just focussing on the protocol specific functionality/features. Feature VIII→X are covered in Chapter 7 and Chapter 9. Chapter 7 focuses on a new methodology for application workload modelling which also provides the ability to provide workload modelling of system calls. ABSOLUT already provides the ABSINTH and ABSINTH-2 workload modelling tools for extracting workloads of user-space code and external libraries as described in (Kreku et al. , 2008b) (Saastamoinen , 2011b). Chapter 9 focuses on the accuracy of the modelled protocols and workload generation methodologies employed by ABSOLUT. Chapter 4 to Chapter 8 cover the description of the models and tools required for providing Features XI and XII. Features XIII and XIV are covered in Chapter 8 which focuses on the modelling and integration of performance probes in ABSOLUT. These probes are used to gather performance statistics during execution. Feature XV is covered in Chapter 6. This chapter describes the way a MOC can be instantiated over ABSOLUT to model a specific domain of applications, for example the modelling of KNP MOC described in this chapter models the streaming multimedia applications very well.

3.4 Summary

We therefore conclude that though a lot of work was done in the area of SLPE, most of the methodologies are targeted to specific system and application modelling domains. Those methodologies which employ a MOC for modelling applications cannot be used for SLPE of distributed networked systems since the MAC and Transport protocols models are absent or abstracted out which play a key role in the end-user experience by contributing to the non-functional properties such as end-to-end delays and packet and frame losses. These non-functional properties must be taken into account for performance modelling of distributed embedded systems so as to ensure a pleasant end-user experience. Though ABSOLUT is agnostic to application or system domain but still it does not contain a lot of features which are important for performance evaluation of distributed systems. In the next sections, we describe the modelling and integration of these features to ABSOLUT briefly. Unnecessary details are avoided and results are mostly not mentioned since all the models, methods described in this thesis and the corresponding results have been published as research articles. These articles provide a detailed description of the modelling style, level of abstraction and tools/languages used in the modelling of these tools and protocols.

4. Multi-threading support for SLPE

In recent years, multi-threaded programming has gained popularity since the general purpose processors have evolved to multi-core platforms. This has resulted in new challenges for software designers in the early stages of the development of multi-threaded applications (both distributed and non-distributed). Therefore, the designers have to do take important design decisions related to load-balancing, thread management and synchronization. This implies that even for moderately complex applications which have a few concurrent threads, the design space will be huge. The exploration of the design space will require the ability to quickly evaluate the performance of different software architectures on one or more platforms. ABSOLUT performance simulation approach has been extended for achieving the faster simulation of multi-threaded applications in the early phases of the design process. Abstract workload models are generated from the source code of the POSIX threaded applications, which are then mapped to the execution platform models for the transaction level simulation in SystemC.

4.1 System synchronization

In case of ABSOLUT, the workload models do not contain timing information. To enhance the simulation speed, the ABSOLUT execution platform models contain cycle approximate timing information. Thus, the platform model dictates the duration of the execution of a particular workload model. The system must ensure the correct behaviour while the concurrent processes are being executed in parallel. This demands a system synchronization mechanism which respects the causal relations between the processes. In other words, this means that any particular execution order of processes or threads is allowed as long as their causal dependencies are respected. From the perspective of posix threads modelling, this can be illustrated via the example shown in Figure 11.

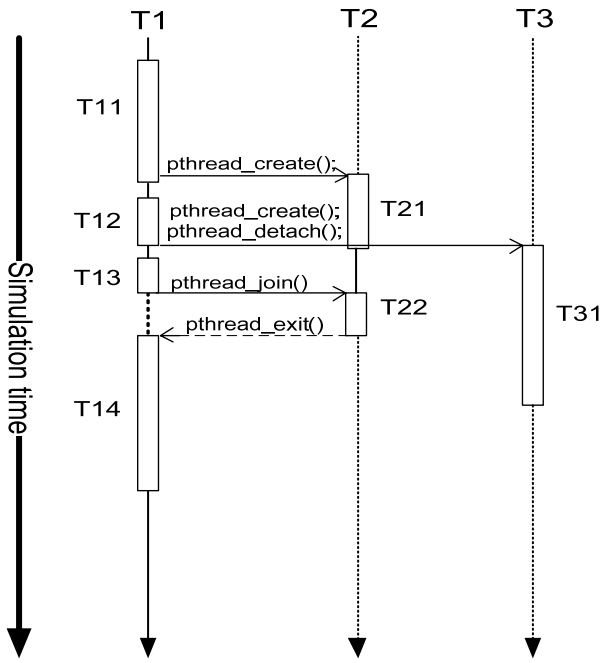

Figure 11: System synchronization between posix threads

We assume that the target the application is following the master-worker programming model. T1 acts as the master thread (main function and is also a workload process). It creates two worker threads T2 and T3 by calling the "pthread_create ()" primitive function. Apart from that, the T3 thread is detached ("pthread_detach ()") i.e., the creator thread (T1) will never block and wait for T3 to terminate.

On the other hand, when T1 calls T2 to terminate ("pthread_join ()), it will block (which is shown via dotted line between T12 and T13) and waits for T2 to complete before it will continue. Both T2 and T3 are independent from each other. The order of execution within each process T1, T2, T3 is as follows

$$T11 \rightarrow T12 \rightarrow T13 \rightarrow T14 \quad (5)$$

$$T21 \rightarrow T22 \quad (6)$$

$$T31 \quad (7)$$

From the correct system synchronization perspective, following are the additional constraints between the processes T1, T2, T3:

$$T11 \rightarrow T21 \quad (8)$$

$$T12 \rightarrow T31 \quad (9)$$

It should be noted that POSIX standard does not place any particular order constraint between T13 and T22 for example, the thread T22 can terminate (reach pthread_exit ()) before T13 calls it to terminate ("pthread_join ()"). In this case, T1 will not block after T13 but will proceed to T14. Following are some examples of the possible overall execution orders in this example:

$$T11 \rightarrow T21 \rightarrow T22 \rightarrow T12 \rightarrow T31 \rightarrow T13 \rightarrow T14 \quad (10)$$

$$T11 \rightarrow T12 \rightarrow T21 \rightarrow T31 \rightarrow T13 \rightarrow T22 \rightarrow T14 \quad (11)$$

Since untimed TLM does not guarantee deterministic execution of concurrent processes, a mechanism for inter-process communication and system synchronization must be integrated to ABOSOLUT platform model. It must also guarantee that the correct intra-process execution order is respected.

4.2 Inter-process communication and system-synchronization model

During the execution of ABSOLUT performance model, the function workloads running on the execution platform model can request different software services by using the service interface called Generic_Serv_IF as shown in Figure 12**.** This interface is realized in the OS model called Generic_Serv_OP_Sys model in Figure 12.

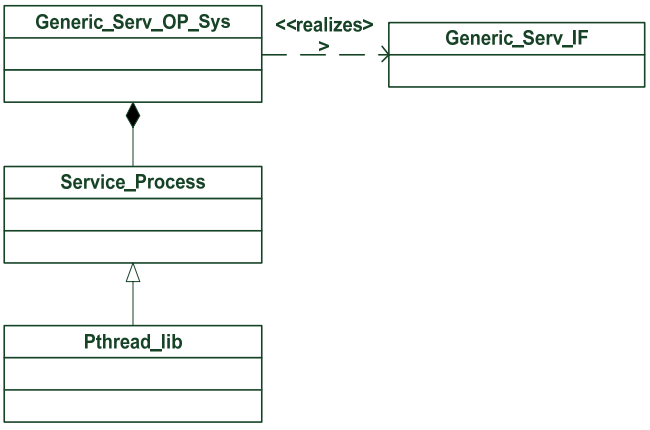

Figure 12: Software service structure

In order to support POSIX service calls from function workload models, a mechanism is needed. We modelled this mechanism in the form of a runtime library service process .i.e., "Pthread_lib" as shown in Figure 12**.** Pthread_lib not only acts as an inter-process communication mechanism but also acts as a thread synchronization layer between the OS model and application workloads as illustrated in Figure 13**.**

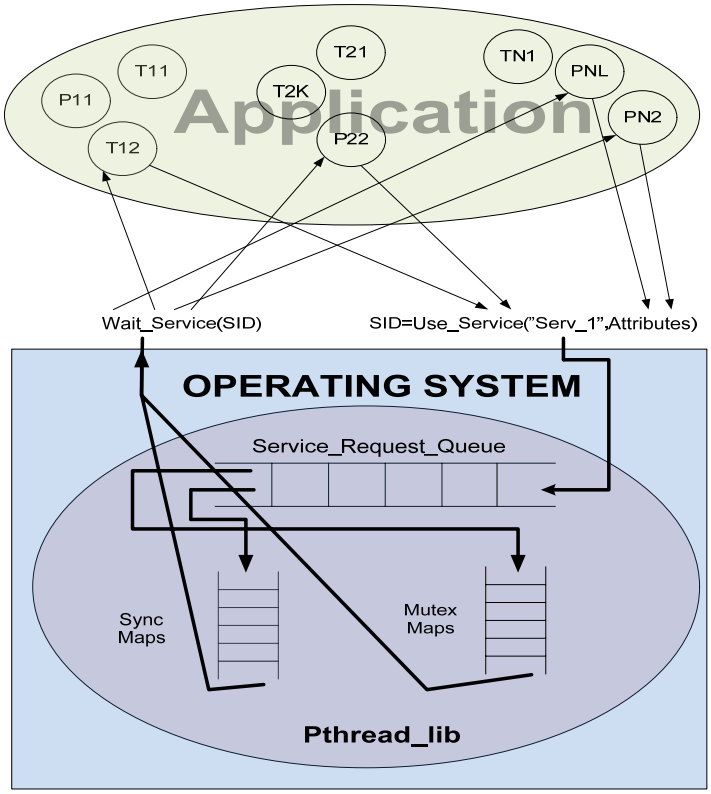

Figure 13: Application workload and OS interface.

A new thread can be created by the function workload by calling the "Use Service ()" call. The service name is used as an attribute to the call, for example Use Service ("Serv_1") along with optional attributes. The unique service name is assigned to a particular service when it is registered to the OS model during the elaboration phase. This is explained in detail in (Khan et al., 2011b). This is a non-blocking request and the calling thread can therefore continue while the service is being processed. The Use_Service () call also returns a unique service identifier which can be given to the blocking "Wait_Service () call to wait for the completion of a requested service.

The OS model relays every new service request to the Pthread_lib service process which puts these requests in the Service_Request_Queue. As the simulation proceeds, a new service request is taken from the queue for processing. Depending on the service name, the "Pthread_lib" object relays the service request to the OS. The OS model schedules the call for execution on the platform. The relaying mechanism can be different and depends on the service type. Further details of the ABSOLUT Multi-threading support modelling and Posix threads along with a complete case study are described in (Saastamoinen et al., 2011).

4.3 Summary

The performance modelling of POSIX-based multi-threaded applications on a high-level general purpose multi-core architecture based processor has been described in (Saastamoinen et al., 2011). A run-time thread service process that ensures correct intra-process execution of application workloads in association with high-level OS model has been described in (Saastamoinen et al., 2011). So far, the correctness of inter and intra-process execution order and compatibility with POSIX API has been emphasized in the development. Therefore, generalization of the methodology to support other parallel programming models like message passing will be one of the focus areas in future development of ABSOLUT methodology.

5. Modelling OS Services (OS_Services) and Background Processes

Extension of ABSOLUT for the performance evaluation of distributed applications requires the modelling of protocols operating at different layers of OSI model. This in turn requires a mechanism for instantiating new H.W and S.W services. These services are registered to the ABSOLUT OS (OS) model and are used by the application workload models. Furthermore, the services operating at a higher layer of OSI model can use lower layer services for example transport-level services such as TCP can use Data-link level services such as IEEE 802.11 MAC protocols for the transmissions of frames of a packet as shown in (Khan et al. , 2011b). These services are instantiated by deriving them from the OS_Service base class as shown in (Khan et al., 2011b). The modelled services implement the Generic_Serv_IF as explained in (Khan et al., 2011b). ABSOLUT functional workload models request the services from the ABSLUT OS model by using this interface. The modelling and integration of highly accurate Data-link and Transport-Level services is explained in (Khan et al., 2011b). The relationship between OS Services operating at transport layer, data-link layer, OS model, OS_Service base class and function workload models is shown in Figure 14.

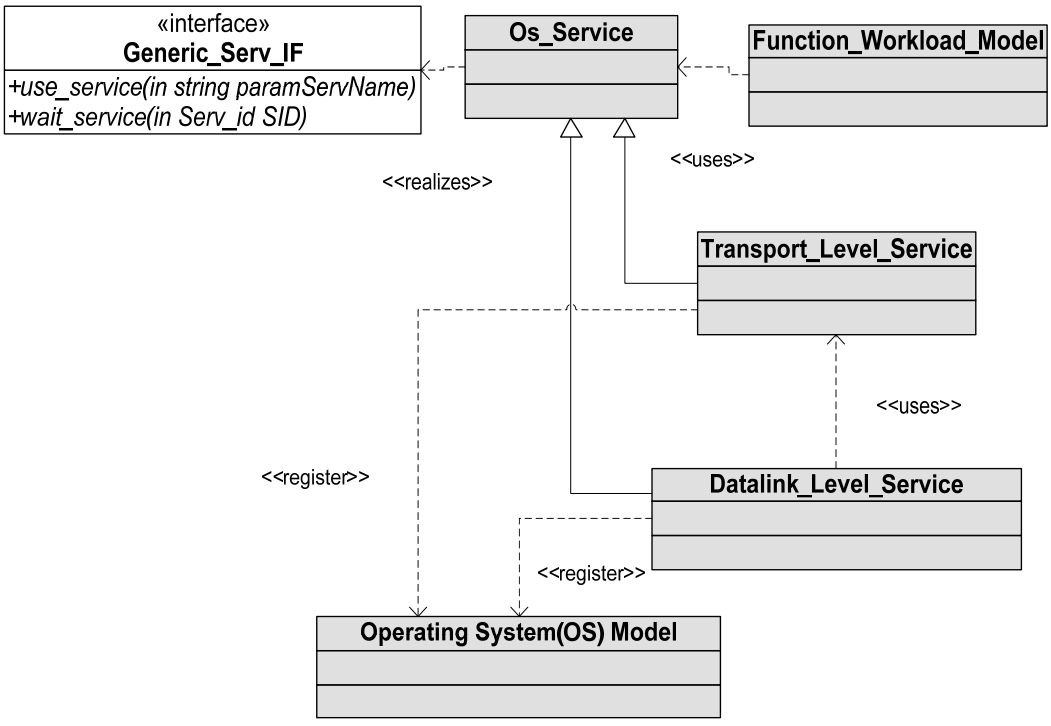

Figure 14: OS services implemented to model use cases spanning multiple devices and for modelling BSD API as OS services.

The OS_Service base class implements the functionality related to the scheduling of requests made by processes via priority queues. After requesting the service from the OS, the requesting process goes to sleep state. The OS informs the requesting process on service completion after which it goes back to the running state. This is shown in

Figure **15**.

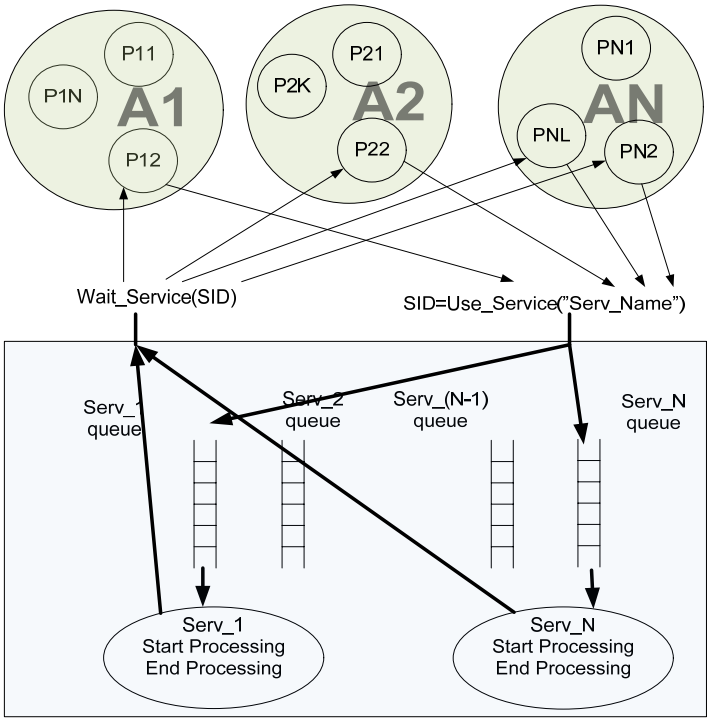

Figure 15: Processing of service requests by OS_Services base class

5.1 Deriving new OS_Services

Only the service-specific functionality is implemented by the derived services which make the modelling of services straight forward. These services are registered to the OS model during the elaboration phase and executed during the simulation when Process or Application level workload models request them from the OS. Other hardware and software services apart from data-link and transport protocols can also be derived from the OS_Service base class. This is shown in

Figure **16**.

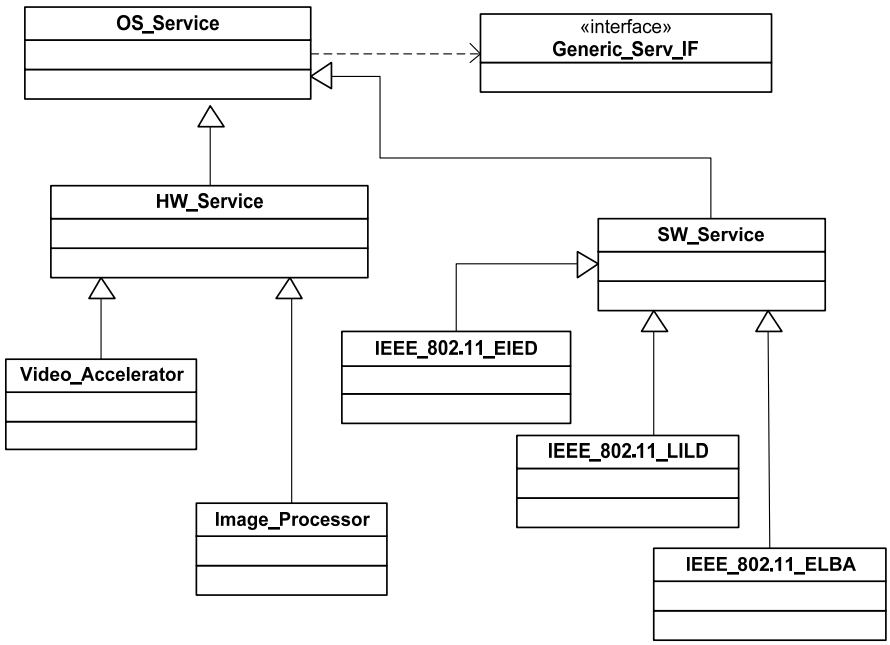

Figure 16: Deriving new OS_Services from the base class "OS_Service"

5.2 Registration and Access of OS services

The services residing at the higher layers of the OSI model use the services at next lower layer of the OSI model in the same way as real world systems for example transport layer services use data-link layer services for transmitting the individual frames of the packets. The services are accessed from the platform using the service name assigned while registration to the OS model. For example BSD socket API function "send()" can be modelled as an *OS_Service* and registered by a unique service name for example "PktTx" to the OS as shown in Figure 17. It can then be accessed by the process workload models by using its unique service name via *Generic_Serv_IF* as shown in Figure 18.

```
//The processor model
ARM_Crtx_Proc_ptr=new Scalable_MultiCore_CPU("m_ARMCortex_A9_MP_Processor");

//Creating the operating system (OS) model
m_os = new Generic_serv_op_sys ("os",ARM_Crtx_Proc_ptr->GetProcessorCores(),m_os_addr);

//Send() function of BSD API registered as "PacketTx" to OS
Pkt_Tx_Service= new Packet_Tx_Serv("Transmit_Packet",m_os);
Serv_type Msg_serv_type = { SERV_TYPE_LOCAL };
m_os->register_service(Pkt_Tx_Service,"PacketTx",Msg_serv_type);

//This service Handles transmission of single frame via IEEE 802.11 DCF registered by the
//name "FrameTx" to OS model
Frame_Tx_Service= new Frame_Tx_ServM3_Msg_Tx("Transmit_Frame", m_os);
Serv_type Frame_serv_type = { SERV_TYPE_LOCAL };
m_os->register_service(Frame_Tx_Service,"FrameTx",Frame_serv_type);
```

Figure 17: Registration of services to the OS (OS) model

As mentioned before, the implementation of the *Generic_Serv_IF* by the *OS_Service* base class enables the function or process level ABSOLUT workload models to request services by their name as shown in Figure 18. This invokes the functionality of that service which is implemented by the service derived from OS_Service base class. The implementation of the OS_Service base class is described next.

```
//Accessing an OS Service
SID=Use_Service("Service_Name");
Wait_Service(SID);
```

Figure 18: Accessing an OS_Service via Generic_Serv_IF

5.3 Modelling of Background Processes

The modelling of workload models for background process is important since many applications running on modern platforms request the services and functionalities from the background processes running on these platforms. Likewise, many middleware technologies such as NoTA DIP are implemented such that they can be used as an external library or as a background process by the applications. In case of daemon mode, NoTA DIP runs as a background process and serves the requests of other processes running on the same platform via the modified NoTA BSD API (Lappeteläinen, Antti et. al., 2008).

The workload models of these API functions are extracted via ABSINTH2 (Saastamoinen, 2011). Nevertheless, the requesting applications cannot execute these function workloads directly. These workload models are executed by the ABSOLUT Daemon models on receiving the corresponding requests (NoTA BSD API function calls) by process workload model of the modelled application. A generic mechanism has been implemented which allows the modelling of daemon/background-process workload models which execute the requests of different processes. This is important since NoTA can also

operate in daemon mode and complex use-cases could also involve daemons and background processes which serve the requests of other processes. The Daemon base class (Daemon_Workload) schedules the requests in a similar way as the OS_Service base class but is also a process in itself in the running state. This is shown in Figure 19. The NoTA daemon process workload model executes the BSP API function workload models extracted by ABSINTH2 (Saastamoinen, 2011) when requested by a process workload model of the AN or SN application model. Other daemons or background processes can also be derived from the Daemon_Workload base class in the same way. Figure 19 follows the UML notation for showing the containment relationship between Application workload model layers.

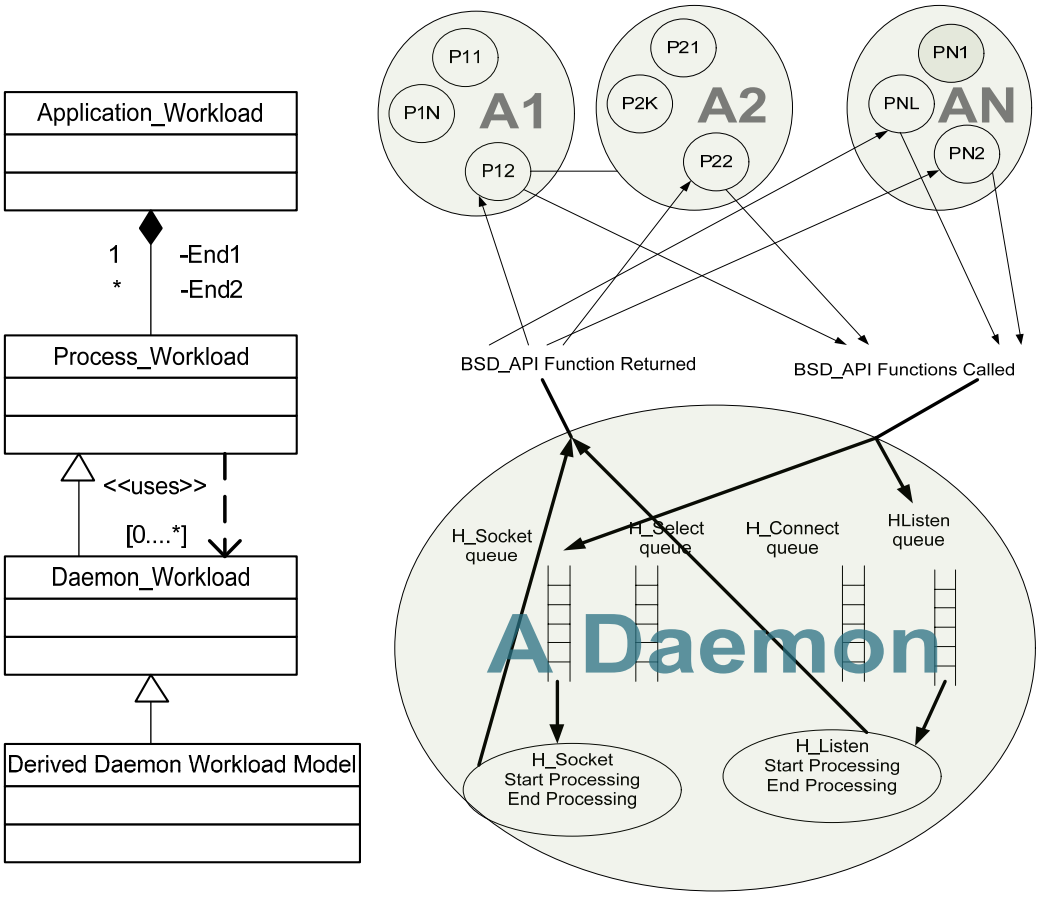

Figure 19: Relationship between daemon workload and process workload (left) and the way it serves the requests of processes.

5.4 Data-link and Transport Layer Services

The Transport Level services .i.e., TCP and UPD etc., and Data-link services for example IEEE 802.11 DCF are derived from OS_Service base class. The Transport level services use the Data-link level services for the transmission of one or more frames of their packets. The IEEE 802.11 DCF operating at Data-link layer can be shown in the form of a flow chart as in Figure 20.

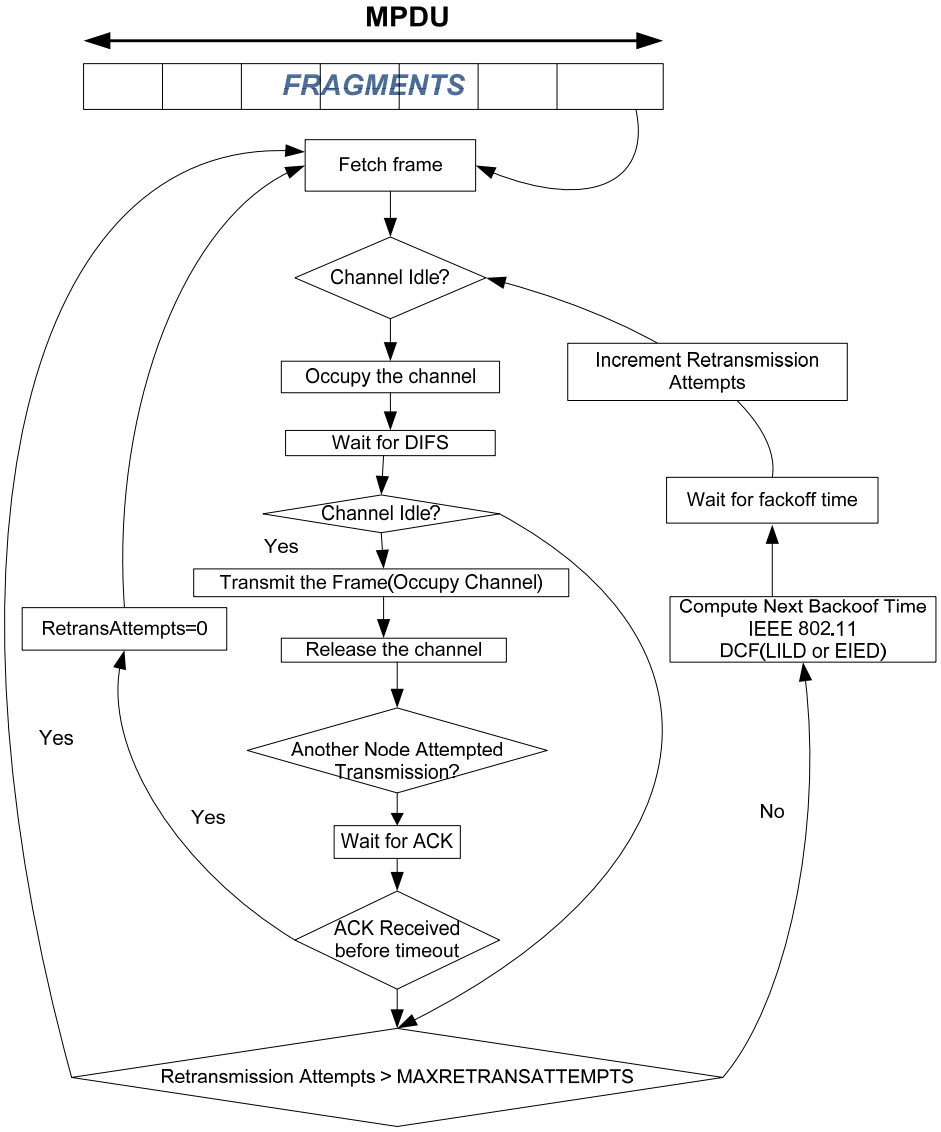

Figure 20: Flow chart of IEEE 802.11 DCF

Every Frame received by the MAC layer is transmitted via the IEEE 802.11 DCF which uses LILD or EIED algorithms for contention resolution as shown in Figure 20. The Transport Layer simply

divides a Packet into Frames and forwards them to The Data-link Layer for transmission. The Data-link Layer stores the Frames in a queue and services them one by one for transmission over the Channel. The recorded simulation results achieve an accuracy of over 92% when averaged after 20 simulation runs when compared to analytical results for Packet Lengths of 228 and 2228 as shown in (Khan et al., 2011b). The packet loss probability achieved an accuracy of over 85% when compared to the analytical results after running the simulation were averaged after running it 20 times as shown in (Khan et al., 2011b). The accuracy of these models is therefore enough for the System Level Performance Simulation of Distributed systems.

5.5 M3 Service and Information Level Services

Seamless access to services and information in a ubiquitous environment can greatly enhance end-user experience. To access this information and services, the embedded devices must resolve the interoperability issues with different mobile devices in the environment. M3 (Multi-Device, Multi-Vendor and Multi-Domain) (Lappeteläinen et al., 2008) addresses this interoperability challenge. It is a possible solution for the interoperability of mobile devices in a ubiquitous environment and divides the interoperability challenge into three levels.

Communication or Device-Level: At the bottom, we have the device world, also called the device level with physical level interoperability and device networks. It provides a capability to transfer bits between the devices. The device or communication level consists of Transport and data-link layers of the OSI models.

Service-Level: At the middle we have the service world also called the service level, where the applications are able to use the services also across device boundaries.

Information-Level: At the top, we have the information world, also called the information level where the interoperability means that the information has the same meaning in different devices. The three levels of M3 are shown in Figure 21. The aforementioned M3 layers are compared to the OSI model in Figure 22.

.

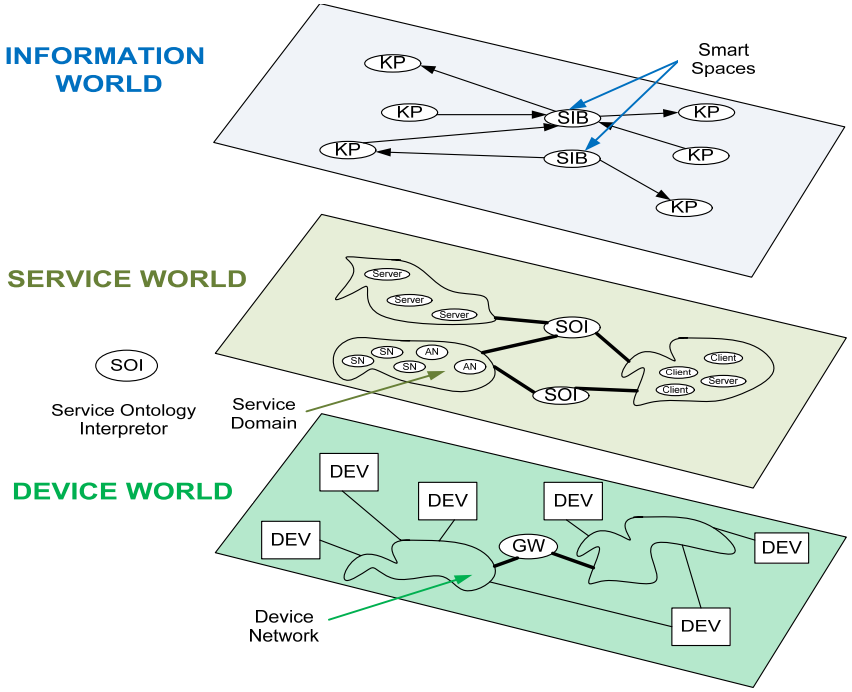

Figure 21:M3 interoperability layers

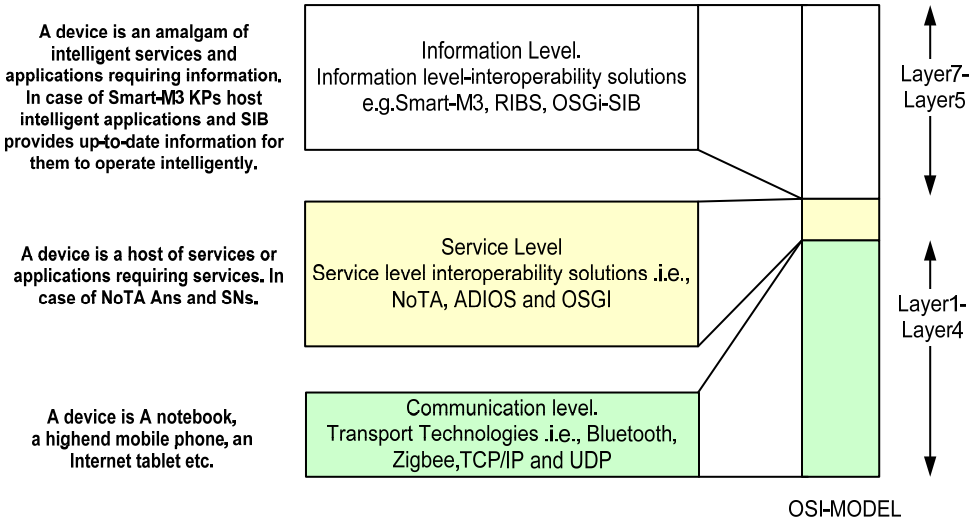

Figure 22: Comparing M3 interoperability layers and OSI model layers

The IOP layers have been explained in detail in (Lappeteläinen, Antti et. al., 2008). The ABSOLUT models corresponding to these layers have already been described in chapter 5.4.We now focus on the way service and information level interoperability platforms/solutions of M3 have been modelled and integrated in ABSOLUT SLPE framework.

5.5.1 M3 Service Level Models

A variety of service level IOP solutions can be employed at Service level in M3. These solutions such as NoTA, OSGI and 2.1.1ADIOS and provide seamless access to services in the Smart Spaces by implementing functionalities related to Service Registration, Discovery etc.,.

NoTA is a M3 service level IOP solution which is available as an external library and as OS services (Kernel implementation). If the external library implementation is used, ABSINTH-2(Saastamoinen, 2011) can automatically generate the workload models of NoTA API functions. Depending on the system designer, the individual API function workloads can be called directly by the process workload models of an application or can be modelled as OS_Services. If NoTA API functions are modelled as OS_Services, they are registered to the OS model during the elaboration phase. During simulation, these services are executed when Process or Application level workload models request them from the OS. Other hardware and software services can also be derived from the OS_Service base class as explained before.

Though we focus on NoTA and ADIOS ABSOLUT models, these service level interoperability platform models can be easily substituted by other Service Level Interoperability (IOP) solutions. Also, apart from modelling Service Level IOP API functions as OS services or external library models, they can also be modelled as the services provided by the background processes. This is particularly useful if the Service Level IOP can also operate as a background process as in case of NoTA (daemon Mode) (Khan et al., 2011c). This is shown in Figure 23.

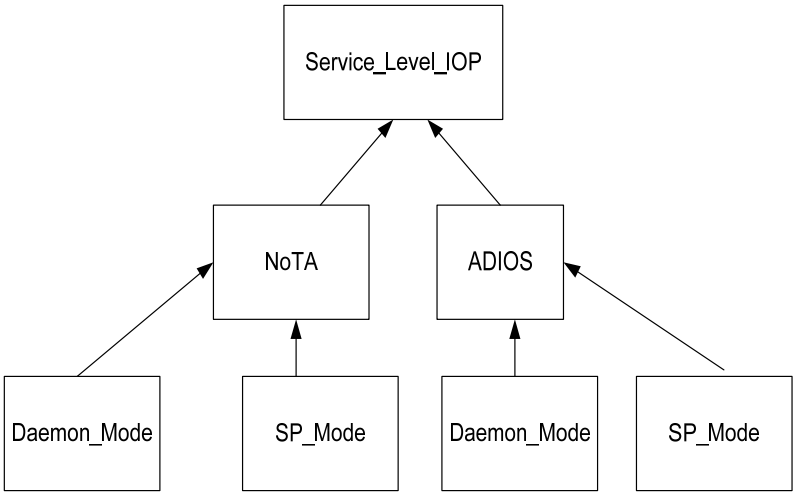

Figure 23. Service level IOP platforms considered and the modes in which they operate.

The relationship between the Daemon and SP workload models of NoTA and the workload models described in ABSOLUT are shown in Figure 24. Figure 24 follows the UML notation for showing the containment and inheritance between Application workload model layers. From this point onwards, all the figures showing application workload models follow the same convention and is therefore not explicitly stated.

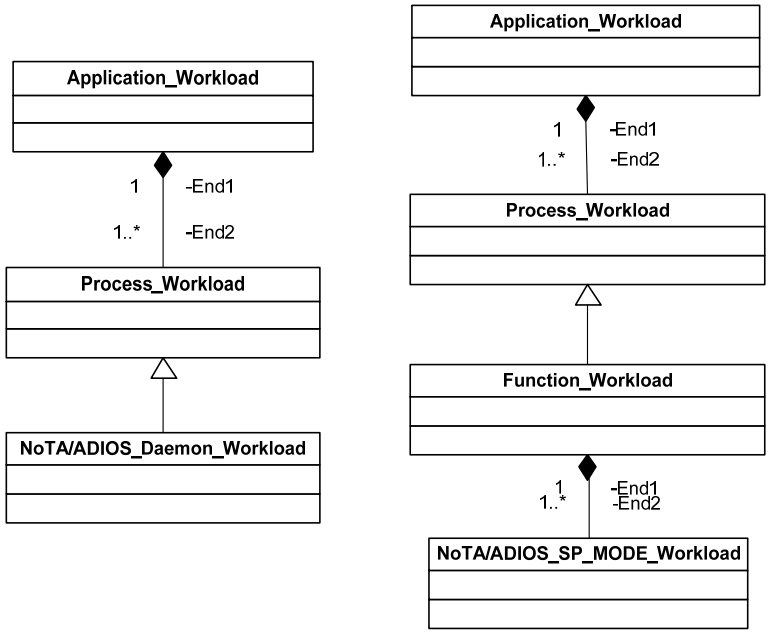

Figure 24. Relationship between Service-Level IOP Daemon-Mode and SP-Mode and ABSOLUT workload models.

When NoTA are used as an external library, it can operate in two modes, .i.e., SP mode and Daemon mode. In case of SP mode, the workload models of NoTA modified BSD API functions are extracted via ABSINTH-2 (Saastamoinen, 2011) and sued by Process-Level workload models in the similar way as normal function workload models ANBSINTH2 (Saastamoinen, 2011).

In case of daemon mode, NoTA runs as a process in the background and serves the requests of other running processes via the same API (modified NoTA BSD API). The workload models of these API functions are extracted via ABSINTH2 (Saastamoinen, 2011). Nevertheless, the requesting applications cannot execute these function workloads directly. These workload models are executed by the ABSOLUT Daemon models on receiving the corresponding requests (NoTA BSD API function calls). A generic mechanism has been implemented which allows the modelling of daemon workload models

that execute the requests of different processes. This is important since NoTA can also operate in daemon mode and complex use-cases could also involve daemons processing the request of other processes. The Daemon base class (Daemon_Workload) schedules the requests in a similar way as the OS_Service base class but is also a process in itself in the running state. This is shown in Figure 19. NoTA daemon workload model executes the BSP API workload models extracted by ABSINTH2 (Saastamoinen, 2011) when requested by a process workload model of the AN or SN application model. Other daemons serving the process workload models can also be derived from the Daemon_Workload base class as shown in Figure 19.

5.5.2 SLPE of M3 applications

The performance evaluation of M3 (at information level) via ABSOLUT consists of the following steps. We assume that RIBS is used as the service level IOP solution and NoTA is used as Service Level IOP solution.

1. The RIBS, NoTA stack and KPs are compiled with the ABSINTH patched GCC compiler and profiling information is enabled.

2. The application is executed. The state of each NoTA stack in a sub-system evolves differently depending upon the functionality of SIBs and KPs that it serves thus different branch probabilities result for each NoTA stack. Also the SSAP Message generation/processing functions in the SIB and KPs generate the profiling information. This profiling information is used by ABSOLUT to adjust the branch probabilities in workload models used in performance simulation (Kreku et al. 2008b).

3. After executing the use-case, the KPs, SIBs and NoTA are re-compiled with the ABSINTH patched GCC compiler (Kreku et al. 2008b). It uses the profiling information gathered during the execution phase and generates a function workload model for each function in the source code of SIB, KP and each NoTA stack residing in each sub-system.

4. The workload models of SSAP_Messages and NoTA stack are mapped to respective platform models and simulated at transaction-level to get performance results. The performance evaluation of Smart-M3 via ABSOLUT consists of four steps as shown in Figure 25.

5. The SIBs and KPs are compiled with the ABSINTH patched GCC compiler and profiling information is enabled. Also the NoTA stack is compiled as in step 1.

6. The application is executed. The state of each NoTA stack in a sub-system evolves

differently depending upon the functionality of SIBs and KPs that it serves thus different branch probabilities result for each NoTA stack. Also the SSAP Message generation/processing functions in the SIB and KPs generate the profiling information. This profiling information is used by ABSOLUT to adjust the branch probabilities in workload models used in performance simulation (Kreku et al. 2008b) Figure 25.

7. After executing the use-case, the KPs, SIBs and NoTA are re-compiled with the ABSINTH patched GCC compiler Figure 25. It uses the profiling information gathered during the execution phase and generates a function workload model for each function in the source code of SIB, KP and each NoTA stack residing in each sub-system.

8. The workload models of SSAP_Messages and NoTA stack are mapped to respective platform models and simulated at transaction-level to get performance results. The function workload models of NoTA BSD API functions are executed from inside the NoTA OS Services or NoTA Daemon workload models if NoTA is modelled as an OS_Service or in daemon mode. If NoTA is modelled as an external library, the NoTA BSD API function workload models are executed from inside the ABSOLUT function workload models or process workload models.

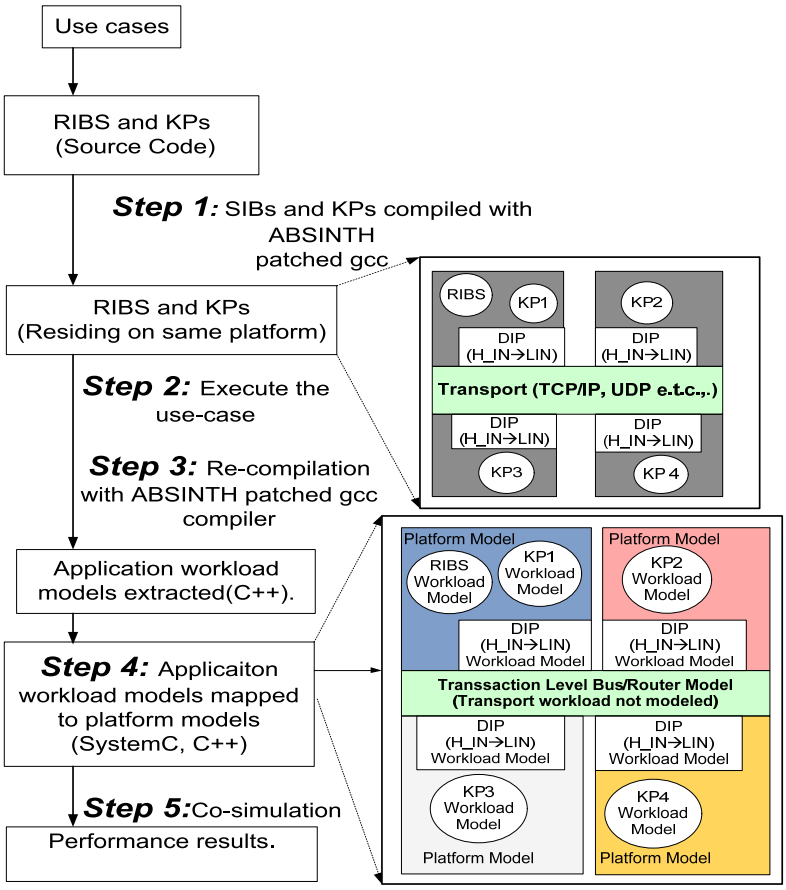

Figure 25. Performance Evaluation of M3 (employing a SOA e.g., NoTA and RIBS) applications via ABSOLUT.

5.6 Summary

Many different use-cases can be simulated by employing the models described in this chapter. These models have a one-to-one correspondence with the real world counterparts .i.e., the functionalities of the protocols/technologies residing at MAC, Transport and other layers of OSI model in real devices are provided by their corresponding models in ABSOLUT framework as described in this Chapter. The models at each layer of OSI model are in the form of OS_Services in ABSOLUT. These models (OS_Services at a particular layer) are made on top of lower layer ABSOLUT models/protocols (OS_Services) at the lower layer. This allows flexibility such that if the system designer wants to retain the protocols operating at a certain layer(s) and wants to modify the protocols at other layer(s), only the layers to be modified are altered. Also, the designer is only concerned with the actual functionality of the protocol rather than its operating system (OS) specific aspects such as the way the service is used by

the application workload models and the way it is scheduled. These OS specific functionalities are implemented in the OS_Service base classes. The simulation results provide valuable insight by identifying the potential bottlenecks in the system at different layers of the OSI model(for each device) and reporting the values of non-functional properties expressed/modelled by the software designer in the software architecture with a high level of accuracy as described in Chapter 9.

6. Instantiating KPN MOC over ABSOLUT

Kahn Process Network (KPN) Model of Computation (MOC) has been widely used by many system level performance simulation methodologies for modelling applications (Mahadevan et al., 2005a, b). KPN MOC models the streaming multimedia applications very well (). When the objective is evaluate the performance of streaming applications, the functional MAC and transport protocols can be abstracted out via KPN MOC. To fulfil the requirements of KPN MOC stated in (Khan et al. , 2011a) previous chapter, the blocking read/write access to FIFO channels, FIFO channels and related services must be implemented and integrated to ABSOLUT (Kreku et al. , 2008b). This chapter represents the way KPN MOC can be effectively adapted over ABSOLUT platform models and used for the performance simulation of streaming applications.

6.1 Properties of KPN MOC

The KPN MOC has a set of properties which must be satisfied to ensure the correct instantiation of KPN MOC over ABSOLUT performance models. These properties are listed below.

a. Read/Write Operations to channels.

The deterministic behaviour of KPNs is mainly caused by blocking reads and writes to the FIFO channel instances.

b. FIFO channel read/write operations

A process cannot wait for reading/writing of two different FIFO channel instances at the same time.

c. Channel Access

If processes can access different FIFO channels and more than one process can run on the same ABSOLUT platform model, then it must be guaranteed that the platform model only allows one process to access a single FIFO channel instance for read/write operation at the same time. This is important since the access to FIFO channels is provided as a service by the ABSOLUT platform (OS (OS) model) to the hosted process workload models and more than one process are allowed to access the same platform service.

d. Process code behaviour

The process code must be blocked of computing while accessing a FIFO channel instance.

e. FIFO channels

FIFO channels cannot be active. In SystemC MOC, it means that the FIFO channel models cannot contain sc_threads or sc_methods.

f. Definition of a KPN processes in ABSOLUT context

The processes of the KPN network formally correspond to software processes. Therefore either the same convention should be followed or the ABSOLUT workload models corresponding to KPN

processes must be identified. In the next subsections, we described the way the aforementioned properties were satisfied by the instantiation of KPN MOC over ABSOLUT. The full adoption of KPN MOC was proved so that the system designers can abstract out certain properties/details if desired.

6.2 Implementing KPN Message Passing Services

Instantiating KPN MOC over ABSOLUT uses the same mechanism for instantiating new platform services (H.W and S.W services provided by the platform) as mentioned in Chapter 5. As stated before, this mechanism is implemented in the form of an OS_Service base class which ensures blocking behaviour and scheduling of the service requests such that only one request is processed at any time for a particular service. The derived services merely implement the functionality making the process of modelling new services straight forward. In this way the required services are easily implemented by deriving them from the OS_Service base class. The token transmission and reception in KPN MOC (via FIFO channels) is also modelled as an ABSOLUT OS Service as shown in Figure 26.

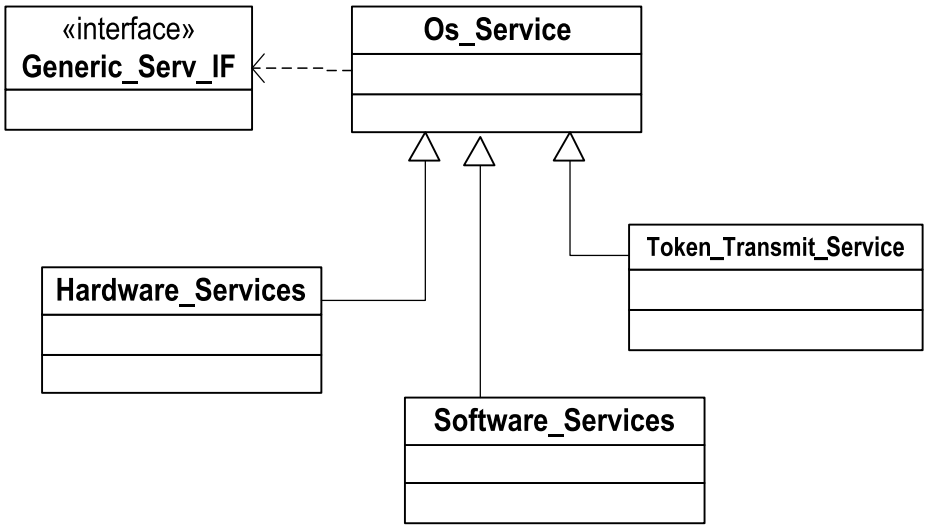

Figure 26: OS_Service base class implementation

As mentioned in Chapter 5, the OS_Service class implements the functionality related to scheduling the requests of processes via request queues and informs the requesting process on service completion after taking it to running state again. At one time only one service request is processed. After the processing of a service is completed, the requesting process is informed and then the next request is taken from the front of the request queue for processing. The requesting process is blocked (remains in the sleeping queue of the OS model) until the execution of the request is completed.

More than one processes running on a single platform can request the same service at the same time (in SystemC it means in the same sequence of delta cycles) and are placed in the service request queue of that service. The implementation of the service processing ensures that only one service is processed at a time and when the processing is completed; the next request is fetched for service processing. This is shown in Figure 27. The figure shows three KPN processes (ABSOLUT process-level workload models) running on the same platform (scheduled by the same OS model) and access the platform services via blocking interface show in Figure 27. Only one request for a particular service is processed at any time and a process cannot request more than one service at the same time since its execution is blocked until the current request is processed which resulted in blocking its execution.

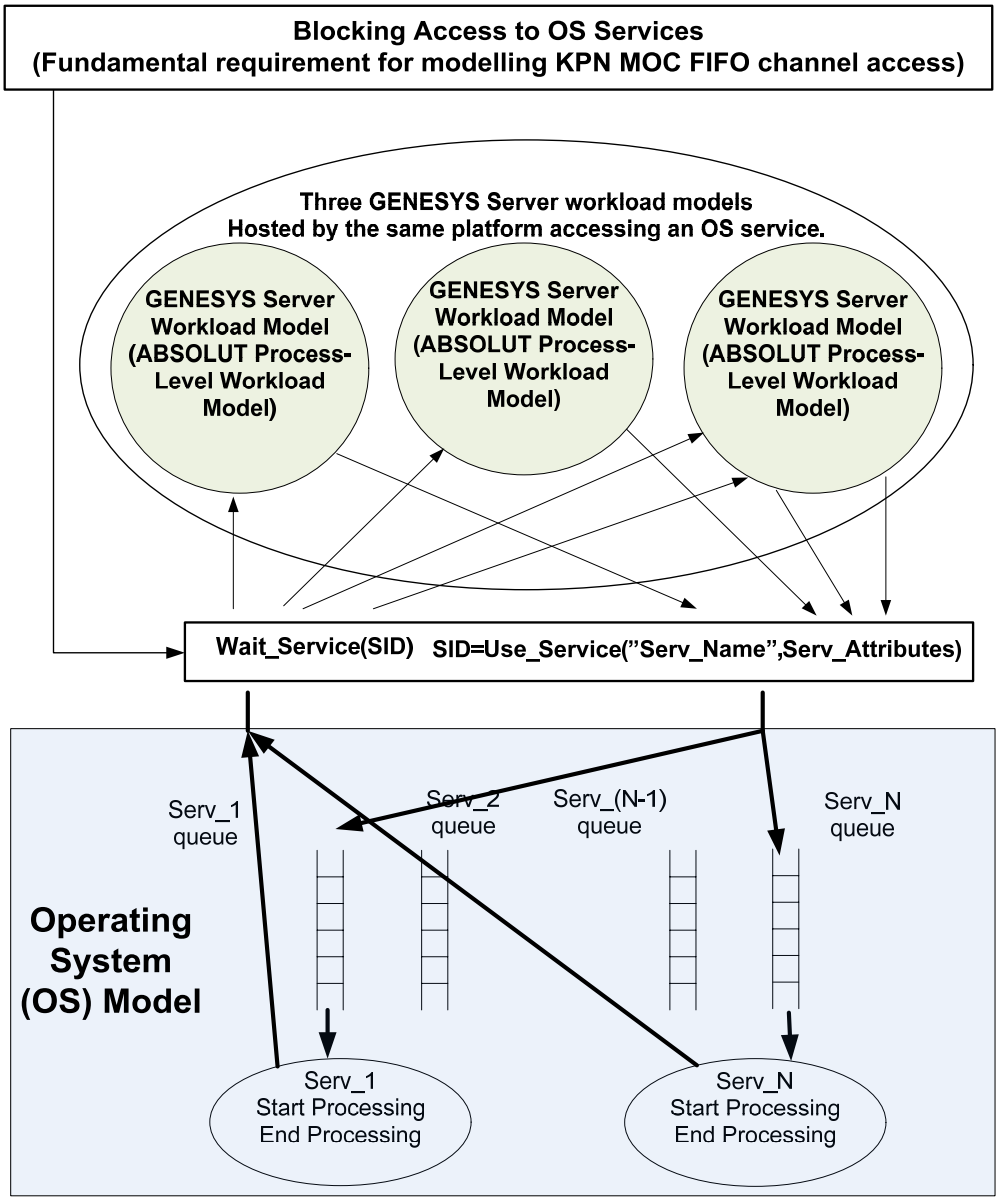

Figure 27: Diagram showing the mechanism employed by OS services to execute requests of processes.

The write access to KPN FIFO channels is implemented as a derived class of OS_Service class and is called “Token_Transmit_Service” as shown in Figure 26. This service is registered to the OS model by the unique service name “TokenTxServ” as shown in Figure 28. The Token Passing service is accessed by the Process Workload models by using its unique service name “TokenTxServ” as shown in Figure 29. The blocking nature of this service (blocks the execution of the requesting process) and the scheduling of service requests via queues ensures that the properties a, b, c and d of the KPN MOC mentioned in Chapter 10.1 are satisfied. The read access to KPN FIFO channels is implemented as a class called “Token_Receive_Service” and is implemented similarly. All the OS_Services are registered to the OS and used via the same interface inside ABSOLUT process workload models.

This implementation guarantees that two or more processes cannot access a single FIFO channel instance at the same time. It also guarantees that one process cannot read and write simultaneously at the same time since the execution of the process is blocked until the request is processed.

The read access to the FIFO channels is also implemented in the same way and is a derived class of the OS_Service base class to ensure that the aforementioned properties (*a, b, c and d*) of the KPN MOC are fulfilled.

```
.
.
.

//Instantiate the operating system model

Generic_serv_op_sys *   ptr_os = new
Generic_serv_op_sys("os",              //The name of OS model
ARM_Crtx_Proc_ptr->GetProcessorCores(),//Multicore CPU cores
m_os_addr);                            //OS address


//KPN token transmitting service registerd to OS model

          //Instantiate the service
Token_Transmit_Serivce * Token_Tx_Service= new
Token_Transmit_Service("Transmit_Packet",ptr_os);

          //Service local to platform
Serv_type Msg_serv_type = { SERV_TYPE_LOCAL };

          // Register servicer to OS
m_os->register_service(Token_Tx_Service,
          "TokenTxServ",Msg_serv_type);
.
.
.
```

Figure 28: Registering KPN Token Passing service

to the operating platform OS system model

```
//Access a service named "Serv_Name" with appropriate attributes

Serv_id SID=SERVICE_OS(m_host)->use_service("TokenTxServ",Serv_Attributes);

//Wait for service completion If it is blocking

SERVICE_OS(m_host)->wait_service(SID);
```

Figure 29: Using Token_Transmit_Service by an ABSOLUT process workload model

6.3 KPN FIFO Channels and Token Modelling

In KPN MOC, the processes communicate via FIFOs channels (Khan et al., 2011a). If a FIFO channel instance is not empty, the reading is non-blocking. If a FIFO channel instance is empty, the reading process will block.

The standard SystemC 'sc_fifo' channel provides these functionalities **Error! Reference source not found.** and we can use them without any modification. The 'sc_fifo' channel has no sc_threads and no sc_methods and hence is not an active channel since activity in SystemC is only modelled via sc_threads and sc_methods. This fulfils the requirement *e* Mentioned in Chapter 10.1.

6.4 Token Passing and Reception

As shown in Figure 29, while requesting an OS_Service, the requesting process must provide the required service attributes. Therefore for accessing the *"Token_Transmit_Service"* and *"Token_Receive_Service"* services, the requesting processes must provide the required service attributes. The attributes for both these services are modelled as a *"KPN_Token_RW_Attributes"* class which is derived from *"Serv_Attributes"* base class.

The *"KPN_Token_RW_Attributes"* class contains a reference to the FIFO channel to which a KPN MOC Token has to be written or read from. Any two ABSOLUT process workload model communicating with each other (running on different devices "platform models") contain a references to the same FIFO channel instance. One of them can only perform read operations on the FIFO channel instance while the other can only perform write operations.

The Tokens do not represent any data of the real use case since ABSOLUT employs non-functional application workload models. Therefore tokens are modelled as integer C++ data type .i.e., int. The 'KPN_FIFO_Ch' class is derived from 'sc_fifo' primitive channel 'class' and does not contain any additional methods or members. The KPN_Token_RW_Attributes class is shown in Figure 30.

```
class KPN_Token_RW_Attributes : public Serv_Attributes
{

//Other class members
.
.
public:

//channel allocation to this token
void Allocate_KPN_Ch( KPN_FIFO_Ch * param_ KPN_FIFO_Ch){
m_KPN_FIFO_Ch=param_ KPN_FIFO_Ch;
}
.
.
//Channel through which to which this token will be passed
private:
KPN_FIFO_Ch * m_KPN_FIFO_Ch;

//Other class members
.
.
};
```

Figure 30: Attributes class for accessing "KPN_Token_Transmit" and "KPN_Token_Receive" service

The aforementioned modelling of KPN FIFO channels and service attributes fulfil the requirement e of the KPN MOC mentioned in Chapter 10.1.

6.5 A KPN process in ABSOLUT context

For seamless integration of distributed GENESYS application design phase to ABSOLUT application workload modelling phase, it is conceived as layered application architecture. After defining the layers in the application model, the corresponding layers in the ABSOLUT workload models are identified. In this way, the application model acts as a blue print for the application workload layers (Khan et al., 2009, 2011a). This reduces the time and effort in application workload modelling and speeds up architectural exploration phase.

In GENESYS, a distributed use-case can be viewed as a controlled collaboration of service providers and service requesters (both called Servers in GENESYS instead of clients and servers) (Khan et al., 2011a) running on different devices. For example, if the use case involves the collaboration among "*n*" GENESYS Servers, we can write.

$E_a = \{ C_E, Serv_1, Serv_2, \ldots, Serv_k \}$, (1)

where C_E represents the control mimicking the collaboration among nodes in order to satisfy the end-user use-case. In the second layer each GENESYS Server is defined as a process running on a par-

ticular platform .i.e.,

Serv = { $P_{GENESYS}$ }, (2)

where $P_{GENESYS}$ represents a GENESYS Server running on a particular platform(called subsystem in GENESYS). In the third layer each running GENESYS server ($P_{GENESYS}$) is represented as a controlled invocation of one or more function workload models or platform service requests. If a process consists of "*k*" processes and "*n*" platform service requests, we can write.

$P_{GENESYS} = \{C_P, F_1, F_2, \ldots, F_k, R_1, R_2, \ldots, R_k\}$, (3)

where C_P is control. The aforementioned GENESYS application model layers are then compared to the ABSOLUT application workload model layers as shown in Table 4.

Table 4: Comparing GENESYS application architecture layers to ABSOLUT Workload model layers

GENESYS Application architecture layers	**Corresponding ABSOLUT Workload Model Layers**
$E_a=\{C_A, Serv_1, Serv_2, .., Serv_k\}$	$W=\{C_A, Servwld_1, ,\ldots., Servwld_k\}$
Serv = { $P_{GENESYS}$ }	Servwld = { $P_{GENESYSwld}$ }
$P_{GENESYS}=\{C_P, F_1, F_2, \ldots, F_k, R_1, R_2, \ldots, R_k\}$	$P_{GENESYSwld}=\{C_P, Fwld_1, Fwld_2, .. Fwld_k, Rwld_1, Rwld_2, .., Rwld_k\}$

Where Servwld is an ABVSOLUT Application workload model, $P_{GENESYSwld}$ is an ABSOLUT process workload model and Fwld is an ABSOLUT Function workload model. Each ABSOLUT process workload model corresponds to a KPN process which invokes the ABSOLUT function workload models and platform services in a deterministic order. Each GENESYS Server Application level workload model instantiates the single process workload model mimicking the execution of a GENESYS server of a distributed GENESYS application in the real use case. Each process workload model ($P_{GENESYSwld}$ in Table 4) corresponds to a single KPN processing node in KPN MOC since the ABSOLUT processes code behaves in the same way as the rules stated in Chapter 5 by using the "Token_Transmit_Service" and "Token_Receive_Service" for FIFO channel access. This observation fulfills the requirement *f* of the KPN MOC mentioned Chapter 10.1.

Therefore from KPN MOC viewpoint, each Application level workload model instantiates a computing node *(in ABSOLUT it means a Process-Level Workload models which are shown in blue color in* Figure 31*)* of the KPN MOC. These computing nodes are connected via unbounded FIFO channels for passing tokens in order to ensure deterministic behavior as explained in (Khan et al., 2011a). Also one or more Process workload models can run on the same platform as in real use cases as shown in Figure 31 and explained in (Khan et al. , 2011a). In Figure 31, two nodes are running on the same platform (Platform 2). Since the access to FIFO channels is blocking and only one process can

read or write to a single FIFO instance at a time as described in Chapter 10.1, therefore the deterministic behavior of KPN MOC is guaranteed.

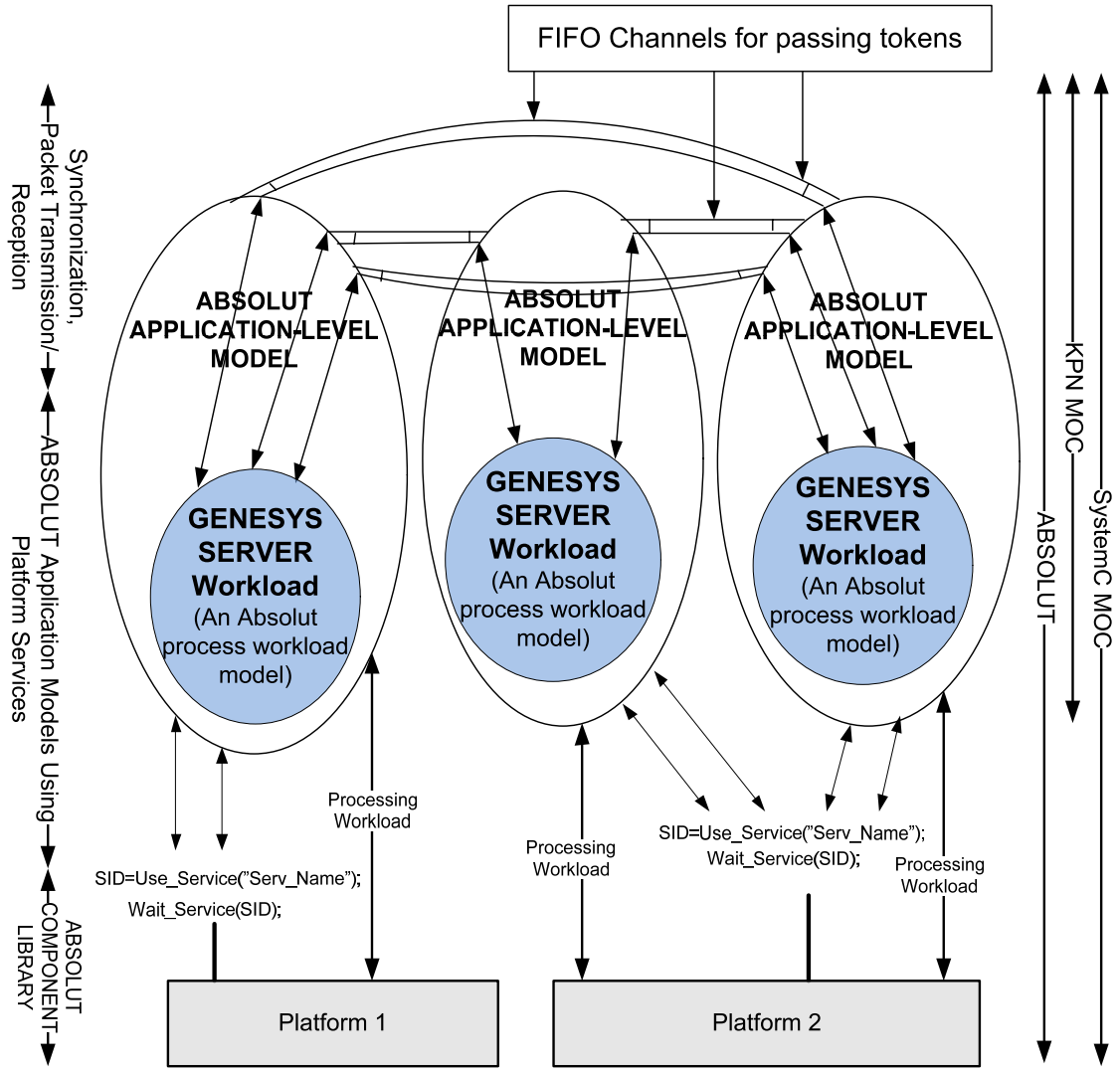

Figure 31: KPN Nodes internals and access to KNP FIFO channels for token passing.

Instantiation of KPN MOC over ABSOLUT is advantageous in those situations where the system designer believes that the contribution of MAC and Transport protocols is minimal or not considered. KPN MOC has been widely used by many SLPE approaches for application design of streaming multi-media applicatins. KPN MOC provides faster simulation speeds due to simple channels and means of communications (Token Passing) among the processes of the modeled application(s). This approach

does not provide a reliable estimate of NFPs such as end-to-end delays and frame/packet loss rate since the OSI model layers contributing to these properties have been abstracted out the simple channel/communication mechanisms employed by KPN MOC.

6.6 Summary

KPN MOC is a good option for the SLPE of distributed embedded systems when the contribution of MAC and transport layers in non-functional properties of the system is not considered. Employing KPN MOC for modelling applications enhances simulation speed due to the simplified communication among application processes via unbounded FIFO channels. KPN MOC has been used by many SLPE methodologies for modelling applications as shown in Table 1. Such methodologies mostly target the domain of streaming multimedia applications.

7. Workload Modelling via Run-Time performance statistics

The automatic application workload modelling tools provided by ABSOLUT previously .i.e., ABSINTH and ABSINTH-2 (Saastamoinen, 2011b).are not able to automatically generate the workload models for system calls. In other words, these methods merely provide the stubs with no workload information for system calls. Therefore, the system designer has to manually provide the workload information in the stubs. Therefore, the accuracy of the modelled workloads for system Calls depend on the experience of the system designer. Furthermore, ABSINTH and ABSINTH-2 are compiler based and might not work with specific version(s) of the compilers. ABSINTH cannot generate the workload models for external libraries while ABSINTH-2 (Saastamoinen, 2011b) uses Callgrind and SAKE (abStract externAl library worKload Extractor) tool for the automatic generation of workload models for external libraries. SAKE tool improves the accuracy of the application models.

Another disadvantage of the ABSINTH is that it models the branches in workload models statistically and, therefore, is sensitive to the quality of profiling data obtained after the execution of application. Hence the higher the level of control in an application, the higher the probability of obtaining highly inaccurate simulation results. The only way to improve the reliability of estimated workload models is to repeat the simulation many times.

The system libraries are pre-compiled and the dynamic libraries are linked to the application at load-time or run-time, therefore the workload extraction for them can only be performed at run-time. Valgrind is an Open Source, Dynamic Binary Instrumentation (DBI) framework that can be used for this purpose. In order to enable ABSINTH to generate the workload models for libraries, the callgrind_annotate tool (a tool for presenting the out of Callgrind (Valgrind), call-graph generating cache and branch prediction profiler) was modified in order to produce the profiling reports that can be post-processed. The SAKE tool is a python script which reads the profiling reports, picks up the calls to external library functions and writes the results to the workload model. Format of the output is by default xml but user can also choose C++ for backward compatibility with ABSINTH flow.

The newly developed methodology described in this Chapter automatically generates the workload models of Systems calls along with external libraries and user-space code. The automatically generated workload models of system calls and external libraries provide the same level of improvement over the manual models as ABSINTH-2 generated workload models for external libraries provide over manual models. Also the application control is not modelled statically and is an exact representation of the run-time behaviour of the application.

7.1 Methodology

The novel contribution presented in this Chapter is an automatic workload extraction and platform processor models configuration method for ABSOLUT called platform COnfiguration and woRkload generatIon via code instrumeNtatioN and performAnce counters (CORRINA) (Khan et al., 2012a). This methodology is completely dependent on the information read from CPU performance counters

and is not compiler dependant. Furthermore, the instantiation of this methodology on a certain platform only requires re-implementation of the interface functions so as to access the CPU performance counters of that machine. The methodology can generate the Application workload models of system calls, user space code and external libraries automatically. Furthermore, it does not employ additional programs like Valgrind for the extraction of workload of external libraries as in ABSINTH-2 (Saastamoinen, 2011).CORRINA is implemented as C++ classes and can be compiled in the form of a static and dynamic library.

The application workload model generation via CORRINA consists of three phases .i.e., Pre-Compilation, Application Execution and Post-Execution phase. In the Pre-Compilation step tags are inserted at different points in the source code automatically via a python script called tag **S**ource-**C**ode pars**E**r written in pytho**N** for CORRINA **T**ags insertion (CORRINA-SCENT) (Khan et al., 2012a). In the Execution Phase, the run-time performance statistics of the application are recorded by reading performance counters for generating the function workload model primitive instructions.

After execution phase, two CORRINA Output files are obtained as shown in Figure 32 apart from the normal output (when the application is not compiled with CORRINA library and no tags are inserted) of the programme. In the Post-Execution Step, the two CORRINA Output files are parsed to generate the classes for Function workload models. Also, the configuration of CPU models is carried out by adjusting cache-hit and cache-miss probability etc., according to the run-time statistics. Also, a top-level process model is generated which calls the generated Function workload models in the order in which they appear in the trace information. The Post-Execution phase is also done via a Python script called CORRINA outPut parsER for FUnction workload generation and process ModEl configuration (CORRINA-PERFUME). The workload modelling and platform configuration via CORRINA is summarized in Figure 32.

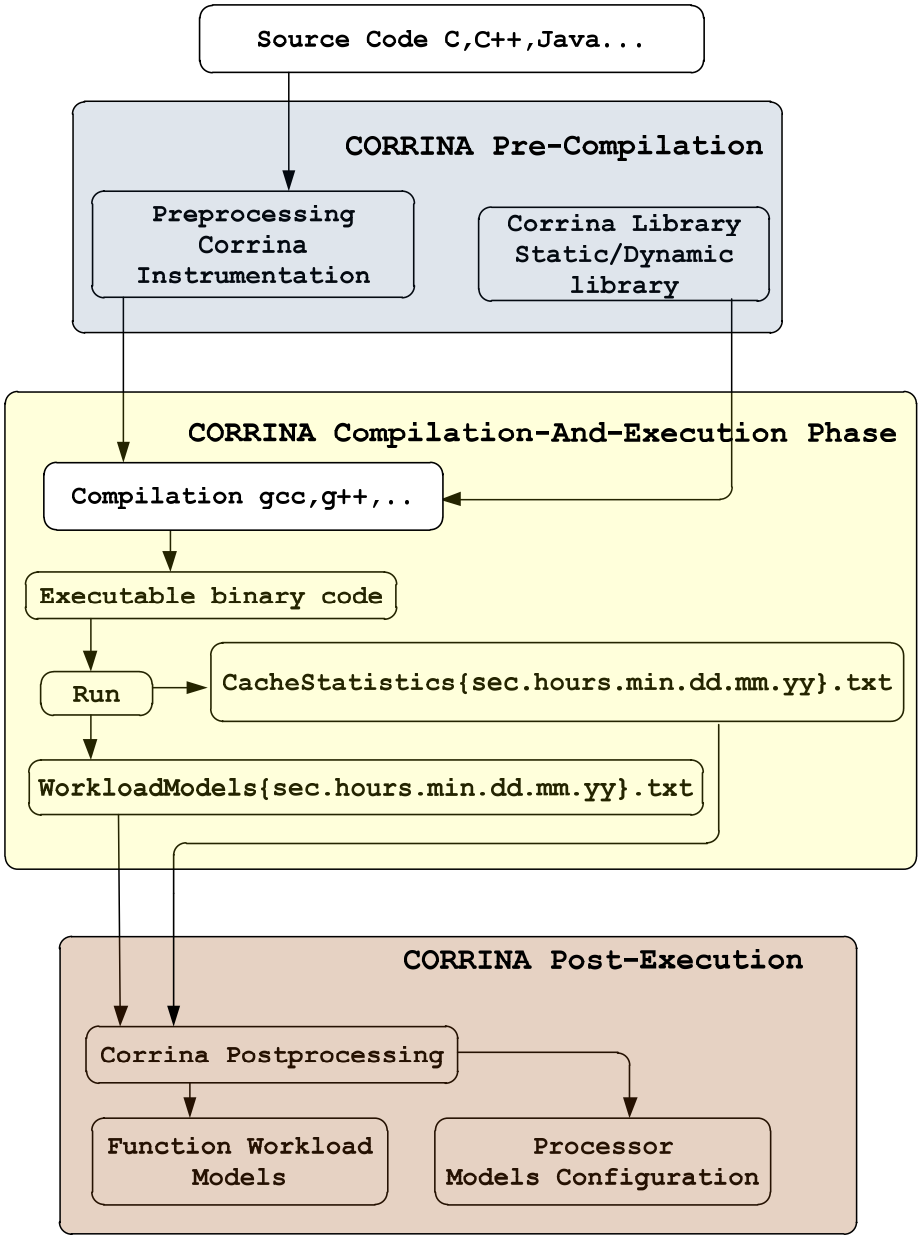

Figure 32: Pre-Compilation and Post-Compilation steps of workload model extraction via CORRINA

As explained before, the workload modelling via CORRINA consists of three phases. The Pre-Compilation Phase, the Application Execution Phase and the Post-Execution Phase. Pre-Compilation phase inserts CORRINA tag-pairs around selected source code lines in the source code. One tag in each tag-pair marks the entry to that code line while the other marks the exit. The second phase is called Application Execution Phase. This phase ends when the application execution ends. During application execution, a separate data structure for function workload model creation is generated for each instruction between tags and a separated data structure for Function workload model creation for the set of instructions encountered along the execution path to the entry of another tag pair, thus giving full coverage of the application source-code. Also the execution order is recorded by recording the names of workload models as they are generated. This trace information is used to call the function workload models in correct order mimicking the true order of execution of the instructions during application execution. In the final step the text files created after the application execution which contains the workload model creation data and CPU models configuration data are parsed and the Function Workload Model Classes generated and the CPU models are configured according to run-time statistics. The aforementioned three phases are described in detail in (Khan et al., 2012a).

7.2 Comparing ABSINTH, ABSINTH-2 and CORRINA

ABSINTH application workload generation methodology is compiler based (Kreku et al., 2008b) whereas CORRINA generates application workload models based on the run-time statistics gathered during the execution of an application. The salient features of ABSINTH and CORRINA are compared below.

a) **gcc Compiler compatibility**

ABSINTH works with certain versions of gcc compiler for example gcc-4.3.1whereas CORRINA is totally independent of the gcc compiler version used to compile the application source code.

b) **Workload of External Libraries**

To extract workload of external libraries, ABSINTH uses Valgrind (Saastamoinen , 2011a). CORRINA does not use any other tool to extract workload models for external libraries.

c) **Workload of System Calls**

ABSINTH is limited to the user-space code and external libraries. Therefore it cannot generate function workload models for the system Calls. CORRINA has no such limitations and the insertion of tags around system calls will provide the required workload models.

d) **Coverage of C++ Applications/g++ compiler**

ABSINITH cannot generate the workload of C++ applications. ABSINTH works as a patch of GNU gcc compiler and does not work with GNU g++ compiler used to compile C++ applications. CORRINA has no such limitations and is not compiler based.

e) **Distributed applications**

Most of the distributed applications communicate via transport technologies such as TCP/IP and UDP. These transport technologies are available to the application programmer as system calls and are used for both message-based communications and also audio/video streaming in case of real-time multimedia client server applications. ABSINTH cannot generate function workload models for these system calls since it is limited to user-space code and external libraries. CORRINA only requires the insertion of tags in the pre-compilation step via CORRINA-SCENT for the generation of Function workload models for these transport API functions.

f) **Processor models configuration**

CORRINA records the cache hits/miss statistics during the entire execution of the application and when the execution of the use-case ends, it writes the overall cache hit and miss rates of the application to the CacheStatistics {sec.hours.min.dd.mm.yy}.txt file for different cache levels. This data is used to configure the CPU models in the ABSOLUT platform model for more accurate SLPE. ABSINTH does not provide this functionality.

g) **Portability**

CORRINA is highly portable but it might require the re-implementation of only the Class member functions used to access the hardware counters in some cases. The complicated tasks of gathering the performance statistics, generation of function workload models during execution, reporting results, pre-compilation and post execution Python scripts .i.e., CORRINA-SCENT and CORRINA-PEFUME don't need to be re-implemented and are totally machine independent.

h) **Application Control**

ABSINTH and ABSINTH-2 model the control statistically which can be highly inaccurate for applications with a high level of control. The control information in application workload models generated by CORRINA are 100% accurate if compared to application execution which resulted in the generated control information.

7.3 Overheads of CORRINA

The overheads of CORRINA are reported in a separate section of the generated output file (Khan et al., 2012 b). CORRINA has two interface functions .i.e., Start() and Stop() as mentioned in (Khan et al., 2012a). The"Stop()" function had a consistent execution time in the range of 13→25 microseconds and "Start()" function only had a consistent execution time in the range of 12→19 microseconds in case of Video Streamer case Study (Khan et al., 2012a). The streamer was working properly and the desired frame rate of 30 Frames per second (Khan et. al., 2011d) was always satisfied. Two other case studies .i.e., QT clock and subattack applications were conducted to monitor the overheads of CORRINA. The Subattack application case study is described in (Khan et al., 2012a).

In case nested tags and recursive tagged functions, the delays are more since due to consecutive occurring of the calls to Start("tag_name") from itself (and same for Stop("tag_name")), the Start() and

Stop() tags which initiate the recursive call chain accumulate the processing times of all inner calls also. This is done deliberately to keep the overheads of performance monitoring of CORRINA itself less. Nevertheless, the overall CORRINA overheads are just a few per-cents which prove the aforementioned point. This fact can be observed in the "Fibonacci case study mentioned in Table 5 which was implemented as recursive function calls. The overall CORRINA overheads with respect the overall application is reported automatically. We used CORRINA with many lightweight applications and still the overall CORRINA overheads were less than 10%. The case studies and the overall CORRINA overheads are shown in Table 5.

Table 5: Overall CORRINA overheads and the corresponding case studies.

Case Study	OVERALL OVERHEADS	Average Execution times of interface functions	
		Start()	Stop()
Clock Application	**.9647%**	**171 µ sec**	**15 µ sec**
Subattack	**4.717%**	**1001 µ sec**	**375 µ sec**
Fibonacci	**1.346%**	**13 µ sec**	**4 µ sec**
Video Streamer	**10.229%**	**15 µ sec**	**15 µ sec**

7.4 Summary

The application workload extraction methods previously employed by ABSOLUT called ABSINTH and ABSINTH-2 had some shortcomings for example these are compiler based workload extraction methods and cannot be used to extract the Application workload models of Kernel space code for example system calls. It lacks the support for g++ C++ compiler and cannot configure the platform processor models according to run-time statistics of the application for example cache-hits/misses (Kreku et al., 2008b). In order to solve these issues, a novel method for workload generation based on run-time performance statistics called CORRINA has been developed which is non-compiler based. Also, CORRINA has some shortcomings since it uses the hardware counters to generate the workload models, the workload models might result in less accurate results when the ABSOLUT platform model has very different hardware architecture.

8. Modelling of Performance probes

In case of distributed networked embedded systems, the delays caused due to the Transport and Data-link layers contribute to the end-user experience and therefore must be measured and analyzed. A mechanism is needed for easy modeling, instantiation and integration of the probes which can record the delays caused by these layers. Due to the wide use of middleware technologies in some domains of distributed applications, it is also important that the mechanism also allows the modeling of probes for measuring delays due to middleware technologies. We next present the modeling of performance probes for the estimation of delays and processing times at different levels of NoTA Applications (Khan et al., 2011c).

8.1 Modelling of Probes for NoTA systems

In order to extract the delays due to technologies employed by NoTA applications .i.e., NoTA DIP, transport and data-link technologies, specialized probes can be easily developed and integrated to a Performance Results class via deriving specialized probes from a base class called Probe. This class performs the functionality that is common to all probes such as storing the recorded delays, producing average results and reporting them to the Performance_Results_Class when the simulation ends. The gathered results are separately reported by this class for each technology or layer targeted by the system designer in the form of a text file. The relationship between the Performance_Results_Class, the Probe base class and the modeled probes is shown in Figure 33. The fugure follows the UML notation to describe the containment and inheritance relationship between different entities.

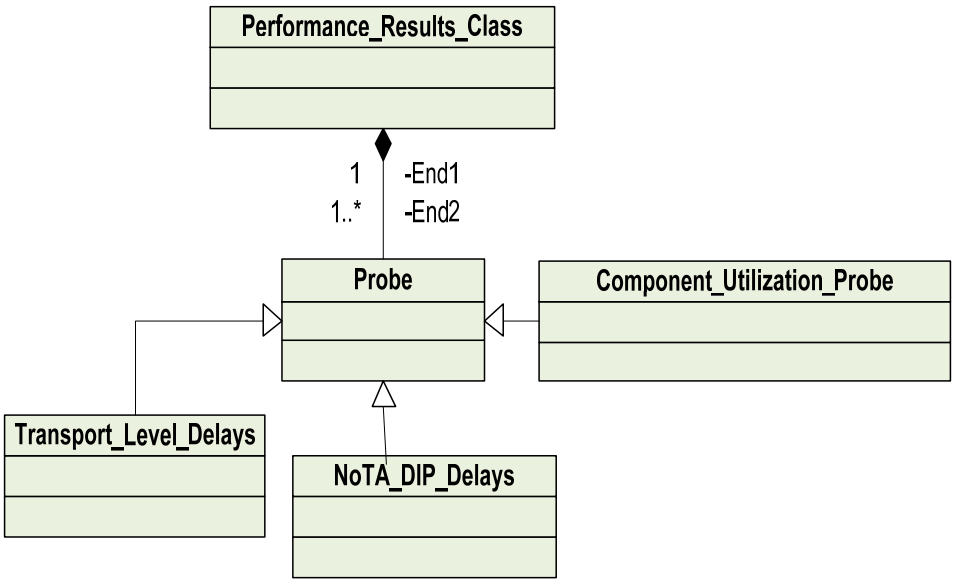

Figure 33. M3 performance results class and probes.

8.2 Measuring Delays and processing times

We define the NoTA API processing times as the average time taken by the target platform to execute the NoTA BSD API functions when triggered or called by an application. These processing times are labeled as T_{NoTA} in Figure 34. The NoTA BSD API functions request services from transport layer, and after performing some tasks for example service access and discovery, they invoke the transport layer functionalities and return. Therefore, the probes recording processing times of NoTA API functions .i.e., $T_{NoTA,\ only}$ measure the processing times of the NoTA BSD API functions .i.e., time taken by the NoTA API functions to return without involving the transport level. This helps to determine the performance of NoTA DIP implementation and the performance of other middleware technologies can also be evaluated similarly. The processing delays measured by the modeled are shown in Figure 34.

We define the end-to-end Device-Level (Transport) delays for connectionless transport Protocols (for example UDP) as the average time taken by the messages as they traverse the sending Smart-Space entity's Transport, Data link (LLC and MAC layers) layers, the physical channel and pass through Data link, and Transport-Layers of the destination device. These delays are labeled as $T_{Transport\_CL}$ in Figure 34.

In case of connection oriented Transport protocols (for example TCP), we define the round trip Transport delays. They are defined as the average time taken by a message as it makes its way from the sending device's Transport, Data-link (LLC and MAC layers), the physical channel and through Data-link, Transport-Layers and Application layers of the destination device and for the message reply to go back through these layers (of the receiver and sender in reverse order) and channel until it reaches the Transport-Layer of the sending Device. These delays are labeled as $T_{Transport\_CO}$ in Figure 34.

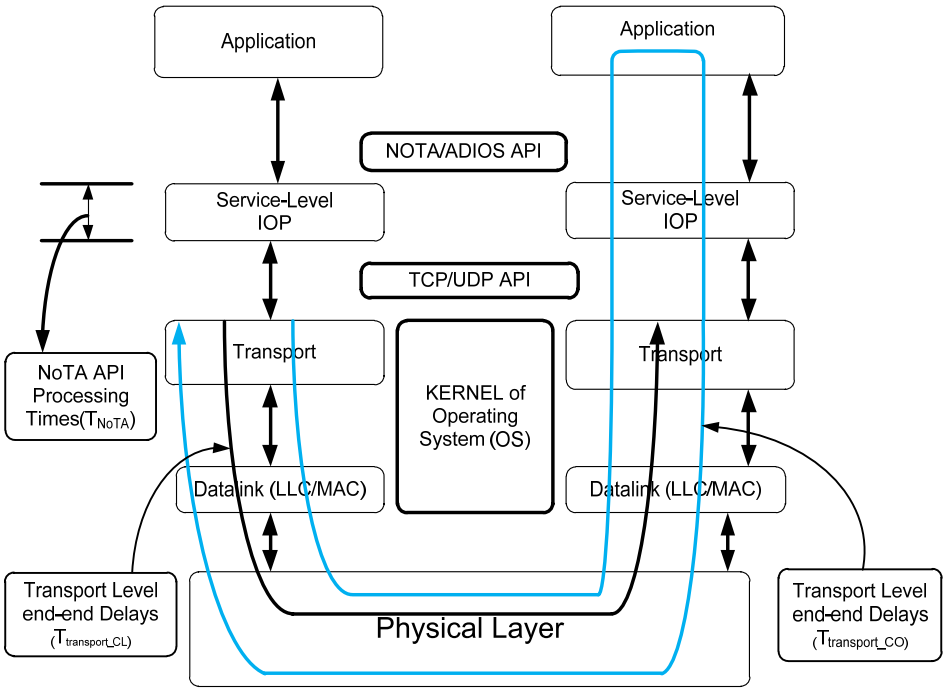

Figure 34: Disgram showing communication between two Smart Space entities. The probes measure Device-Level end-to-end delays and Processing times of Service and Information Level IOP platforms.

8.3 Related Case Studies

The case studies described in the paper III to paper XIII employ the aforementioned method of instantiating probes for measuring the performance of hardware components, MAC protocols, transport protocols and middleware technologies. For recording the performance of platform components, the Component_Utilization_Probe has been developed which records the busy and idle times of a hardware component in the platform as a percentage of the total time taken by the execution of the use-case.

9. Accuracy of Models/Tools

The main objective of the tools/models developed during the research presented in the thesis is to evaluate the performance of distributed embedded systems and applications during the early phases of the system development. The hypothesis was formulated around the following research question.

Is it possible to estimate the potential bottlenecks at different layers in the OSI model (including the application layer as well as the middleware) and other technologies that could be potentially employed by the distributed embedded systems?

After evaluation of different SLPE methodologies in Chapter2 and Chapter 3, it was concluded that ABSOLUT has the potential to integrate the models and tools which will enable it for the SLPE of distributed embedded systems. The extensions made to ABSOLUT are described in Chapter 4 to Chapter 6. These models and tools can only be used for SLPE of distributed embedded systems, if the results obtained by them are accurate enough for taking design decisions. This will enable the system designer to steer the design towards a more optimal design solution which will the non-functional constraints of the system under design. In this Chapter we describe the accuracy of the models/tools integrated to ABSOLUT.

9.1 Software by composition

From an implementation perspective, any embedded system consists of hardware and software components. From an implementation perspective, the software consists of user space code, external libraries and systems calls as shown in Figure 35. Normally the efforts of software developer are mainly confined to the development of user space code which will make use of external libraries/frameworks and system calls to meet the functional and non-functional requirements specified by the customers. Sometimes, even though the functional requirements of the systems are satisfied, considerable time and effort is spent to optimize the system to satisfy the non-functional requirements. Thus, in the context of distributed systems, if the system designer finds the potential bottlenecks at different layers of the OSI model and middleware technologies at an early design stage, it will shorten the design/development time of the system. This requires that the application workload modelling tools should extract the workload models by using different sources, .i.e., application specifications, and source code and run-time statistics.

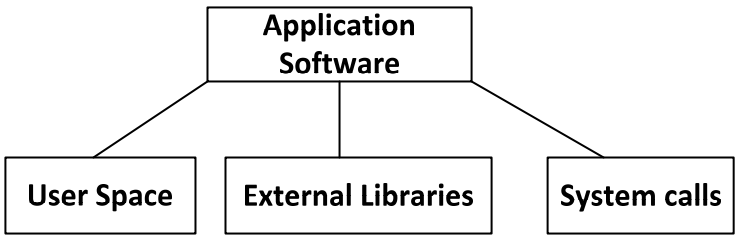

Figure 35: Composition of Software from an implementation perspective

As explained before, the distributed embedded systems might employ different types of technologies. Some only use the OSI model Physical, Network and Transport layer while some use Middleware technologies for brisk application development. Some distributed systems like M3, use higher layer protocols which are built on the top of middleware and transport technologies for providing a specific functionality, for example SSAP in case of M3 systems (Jussi Kiljander et al . , 2011 a,b) . This is shown in

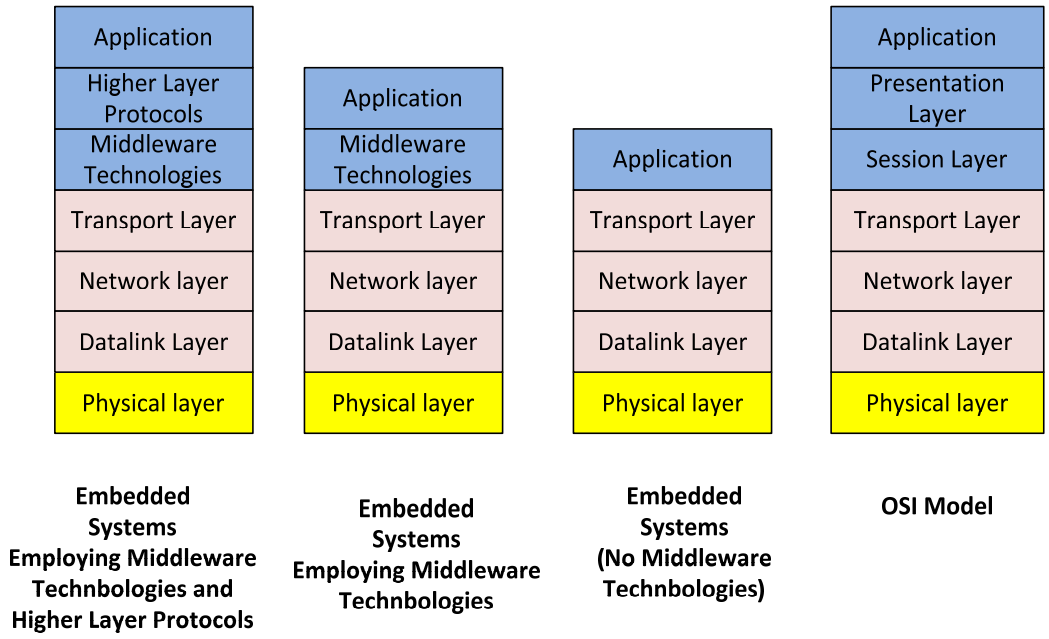

Figure **36.**

Figure 36: Distributed networked embedded systems have evolved, some Use higher layer protocols (such as SSAP in M3), Middleware and Transport Technologies

In the following subsections, we first describe the way ABSOLUT models and tools relate to the implementation of real systems and afterwards we focus on the accuracy of the developed tools and modelled protocols. Chapter 9.2 describes the way ABSOLUT models and tools can be used to model the components of real systems or generate their workload models. Chapter 9.3 focus on the accuracy of ABSOLUT physical layer models. Chapter 9.4 describes the accuracy of ABSOLUT transport and MAC layer models. Chapter 9.5 and Chapter 9.6 describe the accuracy of application workload models and workload models of higher layer protocols.

9.2 ABSOLUT Models/Tools and real systems

After the extensions described in Chapter 4 to Chapter 6, the ABSOLUT methodology can be used for the performance evaluation of different components (both hardware and software) of distributed embedded systems which are shown in Figure 35 and

Figure **36**. We first describe the way ABSOLUT models (OS_Services, background processes models and application Workload models) are related to the components of the real world embedded systems.

We observed that the Transport technologies are mostly implemented inside the OS kernel as System Calls and provide a set of API functions to the application designer for ease of access. Therefore, the Transport technologies can be modelled as ABSOLUT OS_Services. The workload models of system calls can model manually or via CORRINA which is a non-compiler based workload generation method. It is based on run-time performance statistics. CORRINA (Khan et al., 2012a) covers the user-space code and can generate workload models for external libraries also as shown in Figure 37.

It was found that Middleware technologies such as NoTA (Lappeteläinen, 2008) can work in different modes and are available both in the form of external libraries or System calls. Also NoTA can operate as a background process (Daemon mode). Therefore, middleware solutions can be modelled as OS_Services, Application workload models and background processes. If it is available as open source software, ABSINTH is sufficient for workload generation. If it is available as a library, ABSINTH-2 must be used if compiler based workload generation methodology is desired. If the middleware solution is available as system calls, CORRINA can be used for automatic workload generations shown in Figure 37 (Khan et al., 2012a).

The higher layer protocols such as SSAP in M3 are mostly open source and can be modelled as function-layer Application workload models. All the workload generation tools .i.e., CORRINA, ABSINTH and ABSINITH-2 can be used for workload generation as shown in Figure 37. It should be noted that depending on the implementation of these protocols, the ABSOLUT models can be swapped. For example, if a transport technology is available in the form of open source software; it can be modelled as ABSOLUT function-layer application workload models.

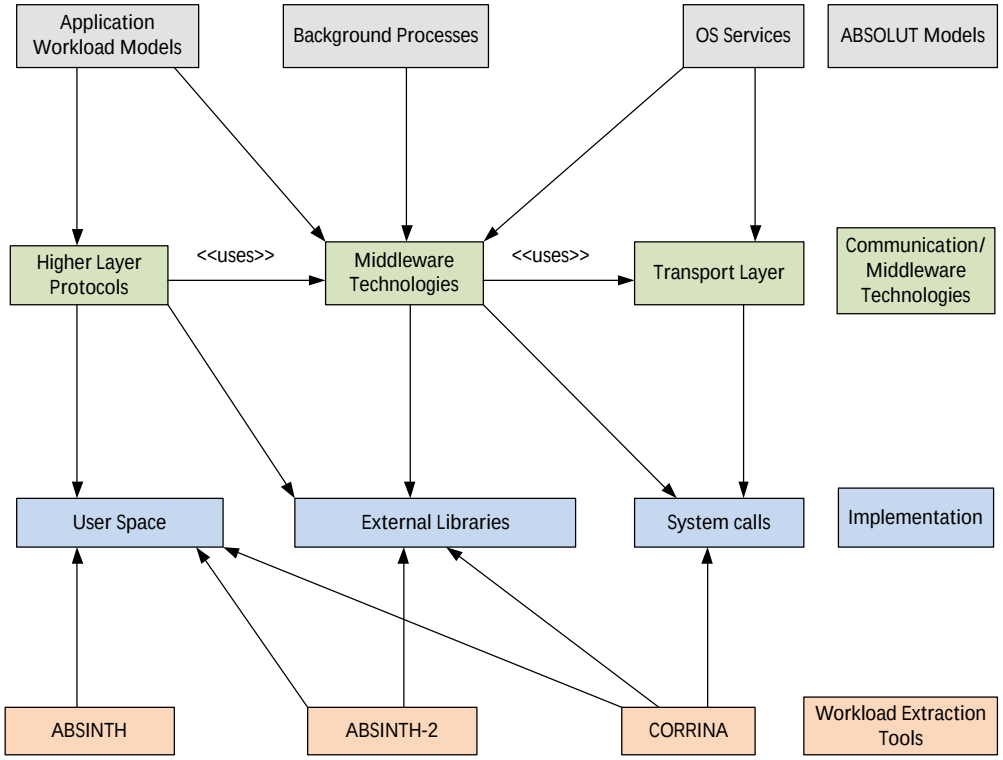

Figure 37: Relationship between ABSOLUT Application models and Software components of distributed Systems. The way different types of embedded software from implementation perspective and ABSOLUT Workload tools

In the next sections, we describe the accuracy of the ABSOLUT models. We first describe the accuracy of Physical Layer models; this is followed by the accuracy of MAC and Transport layer models. In the end the accuracy of Middleware and Application workload models is described.

9.3 Accuracy of Physical Layer models

In order to calculate the bit errors, frame errors and packet errors etc., we employed analytical models for wireless channels, coding techniques and modulation schemes. The models of wireless channels which employ itpp (Khan et al., 2011b). Any other C/C++ library which implements the models of wireless channels can be easily integrated into ABSOLUT framework (Khan et al., 2011b). Also, with very minimal effort the ns-2 and OMNeT++ traffic generator models can be integrated to the framework. So far, the traffic generators of ns-2 have been used for performing the scalability analysis of MAC and Transport protocols in isolation (Khan et al., 2011b). In that case, the application workload models are abstracted out for faster simulation. The calculation of bit and frame errors is also calculated

in the same way as ns-2. First of all, the numbers of bits (frame length) in a transmitted frame MAC frame are used to compute the frame loss probability. This information is further used to compute the packet loss rate etc., as shown in (Khan et al., 2011b). All these calculations are performed under the used channel models, modulation and coding techniques as shown in (Khan et al., 2011b).

9.3.1 Comparing Bit error rate (BER) with analytical models

Different modulation schemes available in itpp library have been used in ABSOLUT for calculating bit error rates (Khan et al., 2011b). We present the results for Multi-Code CDMA with QPSK modulation under simple AWGN channel and without coding. For $1e^6$ bits, the results are over 99.8% accurate (when compared to theoretical results) as shown in Figure 38.

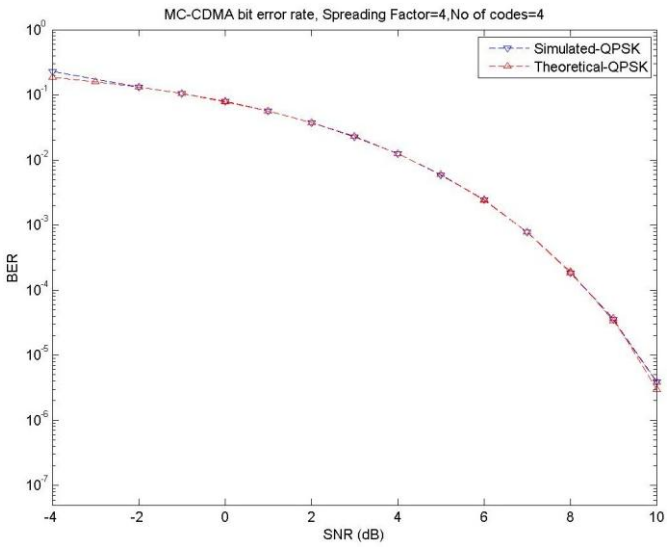

Figure 38: Theoretical versus simulation bit error rate for MC-CDMA with QPSK. Number of codes (M) = 4. Spreading Factor (k=4). Number of bits =100,000.

9.3.2 Comparing Frame error rate (FER) with analytical models

We now present the calculation of FER under the different values of bit error rate to test the reliability of our results. In the absence of any encoding in IEEE 802.11, the fragment and the bit error rate are related by Equation 1 (Khan et al., 2011b).

$$P_e = 1 - (1 - BER)^s \quad (1)$$

Where *s* is the fragment size, *BER* is the Bit Error Rate and *Pe* is the probability of frame error.

The bit error rates are plotted against frame error rates for different values of frame lengths as is shown in Figure 39.

The frame and bit-error rates can be recorded directly from simulation and plotted for different values of bit error rates as shown in Figure 39. The recorded simulation results are over 92% accurate when averaged after 20 simulation runs. The simulation results are compared to analytical results for Packet Lengths of 228 and 2228 as shown in Figure 39. It should be noted that in fact we hardcoded the bit error probability to compute the reliability of frame error probability calculation. In real simulations, the bit error rate values will keep on varying and so will the frame error probability as the simulation proceeds. For example before the frame transmission, the bit error probability is computed from scratch, then frame error probability which helps to decide whether a frame was in error or not. Afterwards, this information is used to compute whether the frame was in error or not. In case, the frame is in error, it is retransmitted by MAC until the retransmission attempts expire. In that case, the frame is discarded (Khan et al., 2011b). This frame loss is reported to transport layer to simulate the packet loss as described in (Khan et al., 2011b). The accuracy of packet loss rate computation is shown in Chapter 9.3.3.

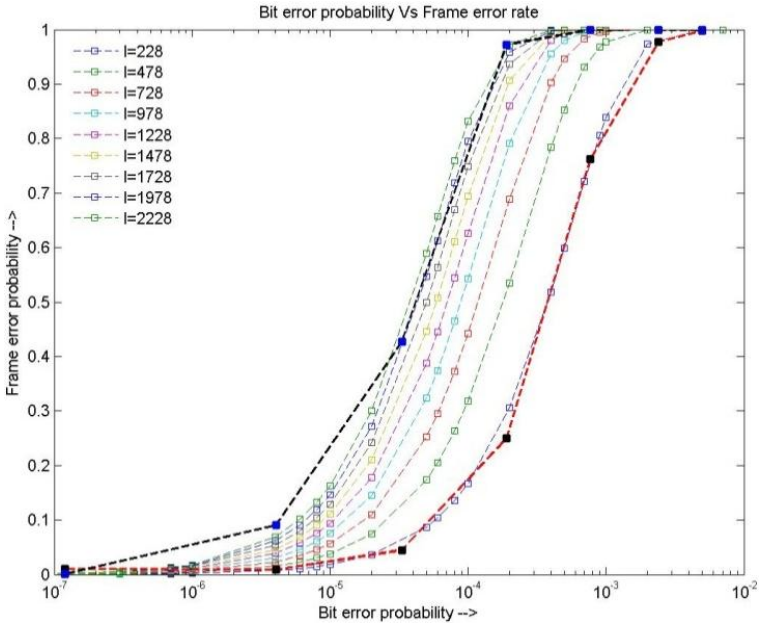

Figure 39: Frame error probability versus bit error rate. Theoretical results compared to simulation results for frame lengths 228 and 2228.

9.3.3 Comparing PER with analytical models

In case of IEEE 802.11, one MAC service data unit (MSDU) can be partitioned into a sequence of smaller MAC protocol data unit (MPDUs) in order to increase reliability. Fragmentation is performed at each immediate transmitter. The process of recombining MPDUs into a single MSDU is called

defragmentation. Defragmentation is also done at each immediate recipient. When a directed MSDU is received from the LLC with a length greater than a Fragmentation-Threshold, the MSDU is divided into MPDUs. Each fragment's length is smaller or equal to a Fragmentation-Threshold (Khan et al., 2011b). The MPDUs are sent as independent transmissions, each of which is separately acknowledged. The loss probability of transmitting a transport packet fragmented at the MAC layer into N fragments is given by the Equation 2 (Khan et al., 2011b).

$$P_{wl} = 1 - \left(\sum_{i=1}^{1=M} P_l^{i-1} \left(1 - P_l\right) \right)^N = 1 - \left(1 - P_l^{M-1}\right)^N \quad (2)$$

Where P_l denotes the successful transmission probability of one attempt, *i* denotes the retransmission attempts and *M* is the maximum number of retransmission attempts. Figure 40 shows the transport packet loss rate as a function of the MAC frame loss probability during each transmission retry for a fixed number of fragments *(N=10)* and for different values of maximum retransmission attempts*(M=1→10)(Khan et al., 2011b)*. The simulation results are compared to the analytical results as shown in Figure 40. The values of *M* and *N* were fixed, the value of signal to noise ratio (SNR) was varied and the simulation was repeated several times. The results for each value of SNR were averaged to obtain each point on the two curves. The simulation was run 20 times and the averaged results achieve an accuracy of over 80% when compared with analytical results as shown in Figure 40.

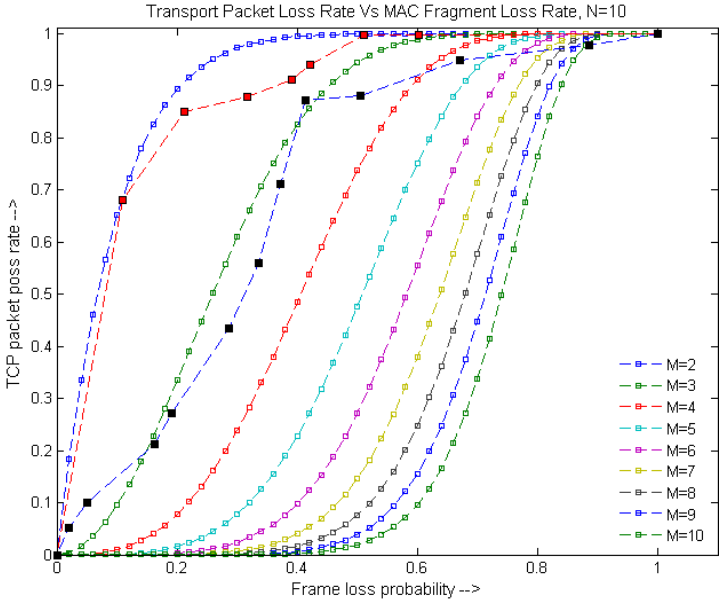

Figure 40: Theoretical versus simulation results. MAC Frame loss probability versus transport packet loss rate, for Maximum Retransmission attempts (M=2 and 3) and number of fragments (N=10).

Therefore, from the aforementioned results, we conclude that the bit error rates, frame error probability and packet loss rates show close correlation with the analytical methods .i.e., over 99%, over

92% and over 80% respectively. We are confident that the correlation will increase if the number of simulations repetitions is increased. These results confirm that the physical layer models (channel models and modulation schemes etc.,) and computation of performance metrics for different layers *(equation 1 and equation 2 in Chapters 9.3.2 and Chapter 9.3.3 respectively)* are accurate enough to be used for SLPE of distributed embedded systems.

9.4 Accuracy of System Call Models (OS_Services)

The System Calls, for example OSI Model's Transport layer API functions and Multithreading API functions (for example posix threads) can affect the end-user experience in two ways. Firstly, they contribute to end-to-end frame and packet delays which are a major concern in a wide range of distributed applications for example streaming audio and multimedia applications. Secondly they can potentially contribute to the platform utilization if the resources of the device are limited or if the implementations are sub optimal. We first look at the way ABSOLUT MAC and Transport models closely mimic the end-to-end delays and throughput of Transport and MAC models of ns-2 and OMNeT++ network simulators and then we describe the way these protocols contribute to the performance of the platform (memories, processors, busses and other digital hardware components).

9.4.1 End-to-end delays and throughput

The end-to-end delays were analysed in the context of Smart-M3 (Lappeteläinen, Antti et. al., 2008). In case of Smart-M3, each network node is called a KP or SIB. Each network node (KPs or SIB) were mapped to a separate ABSOLUT platform model. Each platform model used in the both case studies is a modified OMAP-44x platform model. The MAC and Transport services were registered to the OS model of the platform. The platform model consists of two ARM Cortex-A9 processors consisting of four and three processing cores respectively instead of two (as in case of original TI OMAP44-x platform , SDRAM, a POWERVR SGX40 graphics accelerator and an Image signal processor as shown in Figure 48. The Network-on-Chip (NoC) infrastructure is abstracted out and replaced with on-chip bus as shown in Figure 48.

We now compare the throughput, Frame delays and Packet delays of ABSOLUT MAC and Transport models with ns-2 and OMNeT++ network simulators. Different Packet lengths, transport protocols and Packet transmission rates are used in both the case studies. The simulations are carried out under saturated conditions. The simulation parameters are mentioned in Table 6.

Table 6: Experiment parameters

Parameters	Values
SIFS	10 micro seconds
DIFS	50 micro seconds
Slot Interval	20 micro seconds
Preamble Length	144 bits

PLCP header Length	48 bits
Channel bit rate	2 Mbps
CWmin	32
CWmax	2048
CWo	32
EW	16

The simulations are carried out in WLAN environment in the context of M3. The abbreviation M3 means multi-device, multi-vendor, multi-domain to highlight the flexibility and portability of the technology (Lappeteläinen, Antti et. al., 2008). It means that all the network client nodes called Knowledge Processors (KPs) at information level in M3 are within the transmission range of a single server called Semantic Information Broker (SIB) which acts as the only destination for the KPs.

9.4.1.1 Case study 1: Comparison with ns-2

In the first case study we compare the results of our MAC and transport models with ns-2. The data traffic is generated using the Constant bit rate (CBR) traffic generator available in ns-2 simulator and the transport protocol is TCP (Khan et al., 2011b). The traffic generators can be configured by using the interface shown in Figure 41.

```
//CBR traffic generator parameters

// 2.5 milliseconds
double AverageBurstTime=.0025;
// 1 second
double InterFrameTime  = 1   ;
// 2048 bytes
double SinglePktSize=2048;
// 2 Mega bits per second
double Bitrate=2000000;

//Instantiate CBRtraffic generator
SimConfig::Instance()->SetTrafficGenerator(
CBR_TRAFFIC_GENERATOR,
AverageBurstTime,
InterFrameTime,
Bitrate,SinglePktSize
);
```

Figure 41: An example configuration of the CBR traffic generator. Packet Length=2048 Bytes. Data rate= 2 Mbps. Average Burst Time=.0025 seconds. Inter Frame Space=1 second.

With the same experimental parameters mentioned in Table 6, we perform our simulations in two different frame lengths .i.e., 512 bytes and 1024 bytes. In both the cases, the transport-level packets are

not fragmented into multiple MAC frames; therefore we only use the MAC Frame delays and throughput for comparison. The average Delays for both the frame lengths are shown in Figure 42 for different number of active nodes (20→100 KPs and one SIB in smart spaces) in the network (Smart Space).

The ns-2 and ABSOLUT simulations were run 50 times and the average values were computed. The results indicate that if ns-2 is used as a reference bench mark, the results of ABOLUT Mac and transport are 70-80% accurate. The inaccuracy is due to absence of the RTS/CTS mechanism in ABSOLUT models. The results show that ABSOLUT models always produce pessimistic results, .i.e., less throughput and more delays for the same simulation scenario.

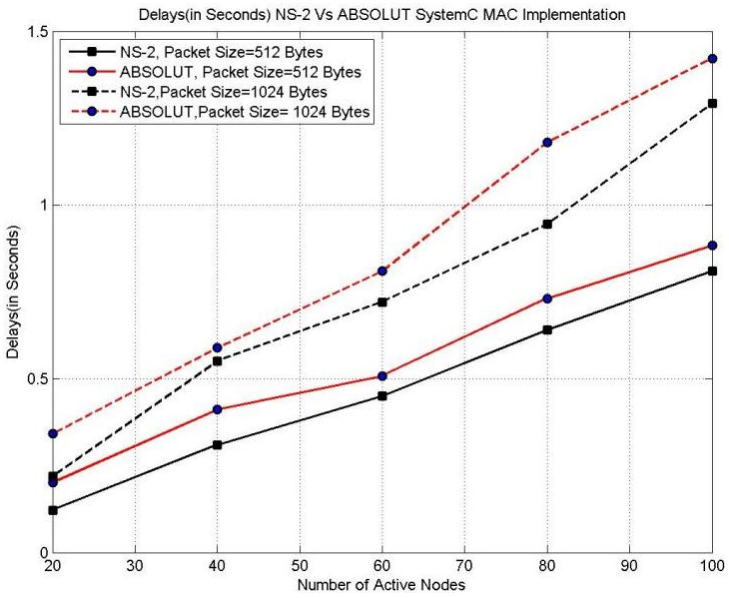

Figure 42: Delays (seconds) vs. number of active nodes

(Ns-2 versus ABSOLUT)

The normalized throughput for both the frame lengths is shown in and Figure 43.

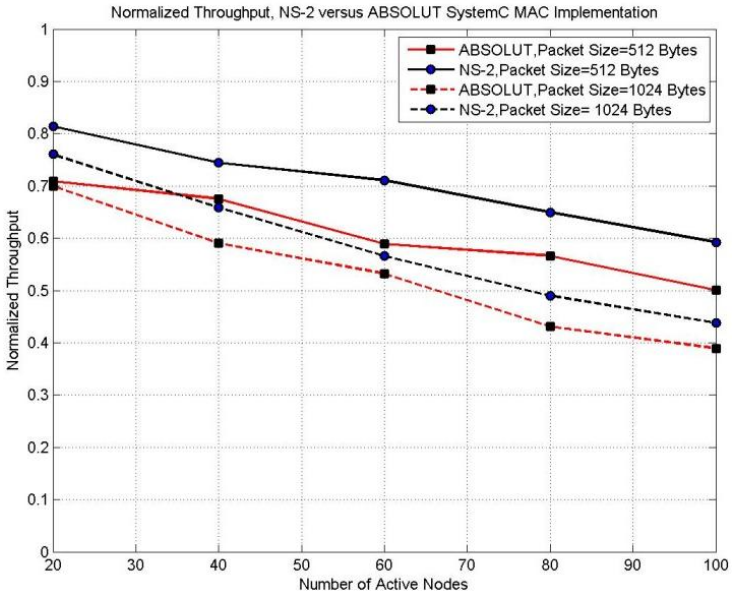

Figure 43: Normalized Throughput versus number of active nodes (Ns-2 vs. ABSOLUT)

The average collisions times (Number of collisions/100 seconds) are shown in Figure 44.

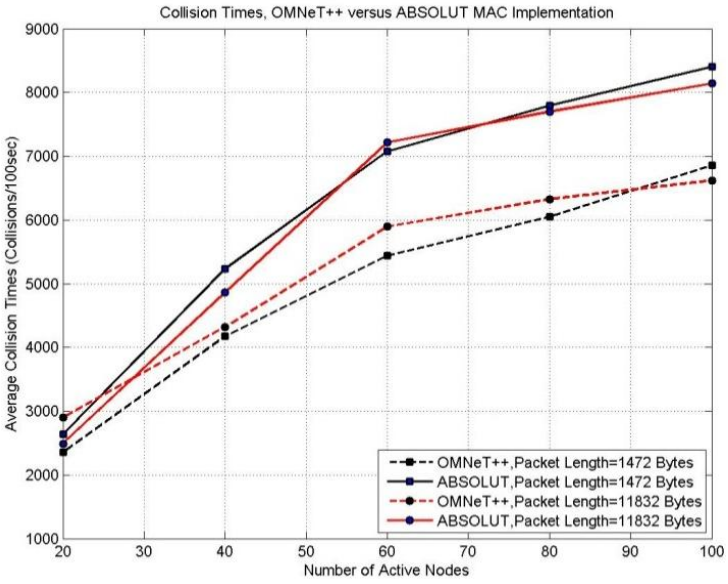

Figure 44: Average collision times vs. number of active nodes (Ns-2 vs. ABSOLUT).

9.4.1.2 Case study2: Comparison with OMNeT++

In the second case study, we compare the results of ABSOLUT MAC and transport models with OMNeT++. No traffic generators were used. The application sends packets at 2 millisecond interval. The simulations are performed under two scenarios. In the first scenario, the application sends 11832 Bytes long packets. The packet is fragmented into 8 fragments. As a consequence MAC sends 8 fragments, each of length 1500 Bytes. In the second scenario, the application sends a 1472 Byte packet at 2 millisecond interval. There is no fragmentation on any layer and as a result, MAC sends a single frame of length 1534 Bytes for each Application packet. Since the packet transmission rate is too fast, the collision rate is quite high which significantly increases the delays and reduces the throughput.

The maximum and average Delays for the packet length of 11832 Bytes(8 Frames/Packet) is shown in Figure 45 as the number of nodes (KPs) is varied (20→100) in the network (Smart Space).The goal is to investigate the case where multiple frames are transmitted for a single transport packet.

The OMNeT++ and ABSOLUT simulations were run 20 times and the average values were computed. The results indicate that if OMNeT++ is used as a reference bench mark, the results of ABOLUT MAC and transport are 75-90% accurate. The inaccuracy is due to the absence of the RTS/CTS mechanism in ABSOLUT models. The results show that ABSOLUT models always produce pessimistic results, .i.e., less throughput and more delays for the same simulation scenario.

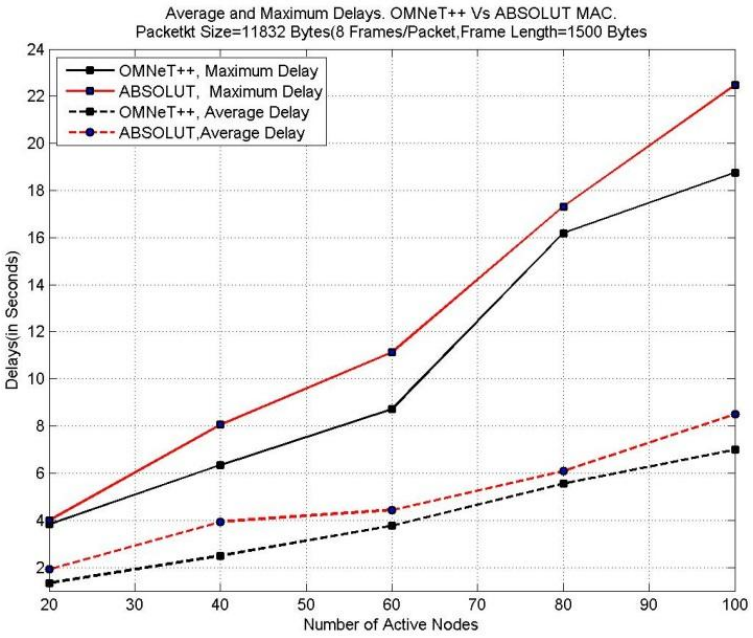

Figure 45: Maximum and Average Delays (seconds)

Vs. number of active nodes (OMNeT++ vs. ABSOLUT).

The normalized throughput for both the packet lengths is shown in Figure 46.

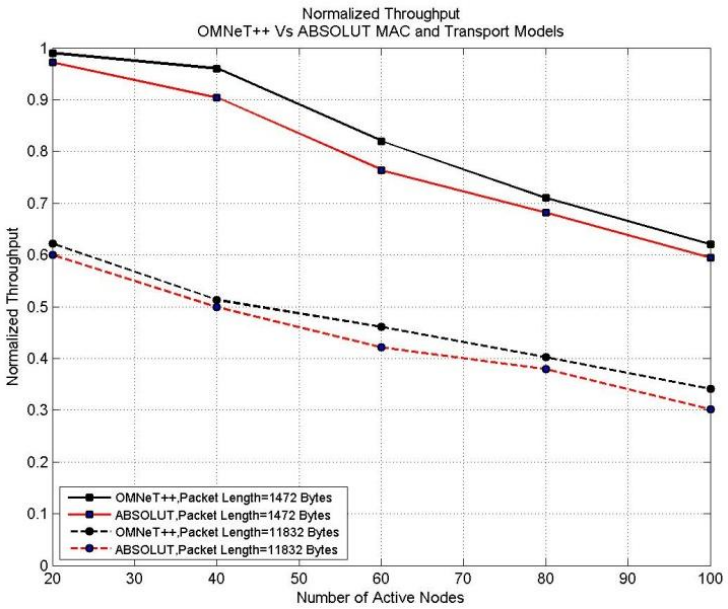

Figure 46: Normalized Throughput Vs. number of active nodes (OMNeT++ Vs. ABSOLUT).

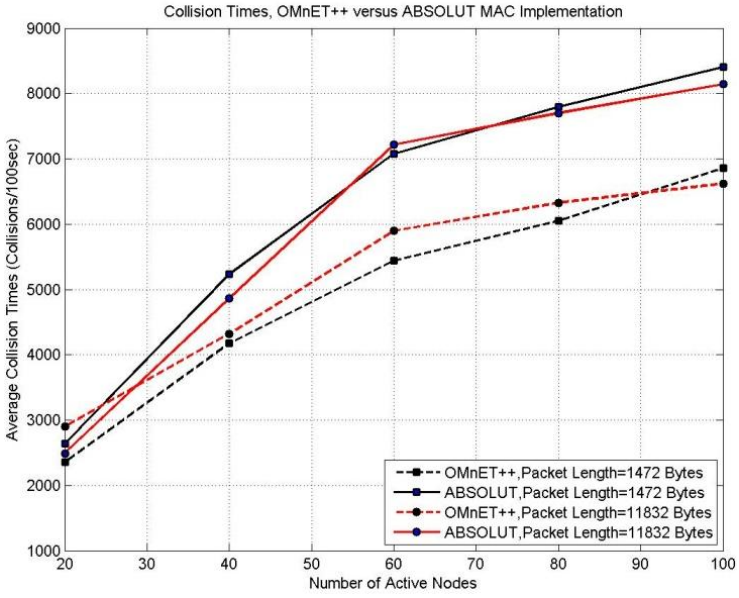

Figure 47: Average collision times vs. number of active nodes (OMNeT++ Vs. ABSOLUT).

9.4.2 Platform utilization and Contribution to Overall Workload

Each network node (KPs or SIB) were mapped to a separate ABSOLUT platform model. Each platform model used in both case studies was a modified OMAP-44x platform model. The MAC and Transport services were registered to the OS model of the platform. The platform model consists of two ARM Cortex-A9 processors consisting of four and three processing cores respectively instead of two (as in case of original TI OMAP44-x platform), SDRAM, a POWERVR SGX40 graphics accelerator and an Image signal processor. This platform model and the quad-core processor model are shown in Figure 48 and Figure 49. The Network-on-Chip (NoC) infrastructure was abstracted out and replaced with on-chip bus as shown in Figure 48.

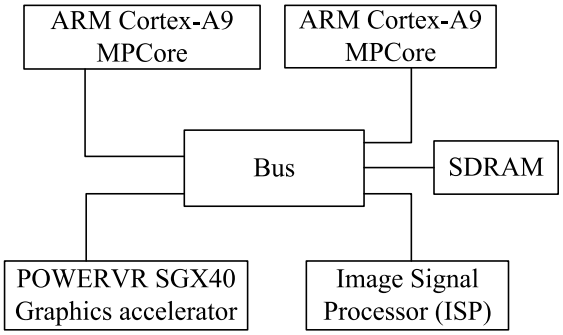

Figure 48: OMAP 44x Platform ABSOLUT model.

Each processor core (Cortex-A9 CPU model) has an L1 and L2 cache and can possibly share an L3 cache with one or more cores in the Multi-Core Processor model. This is shown in Figure 49.

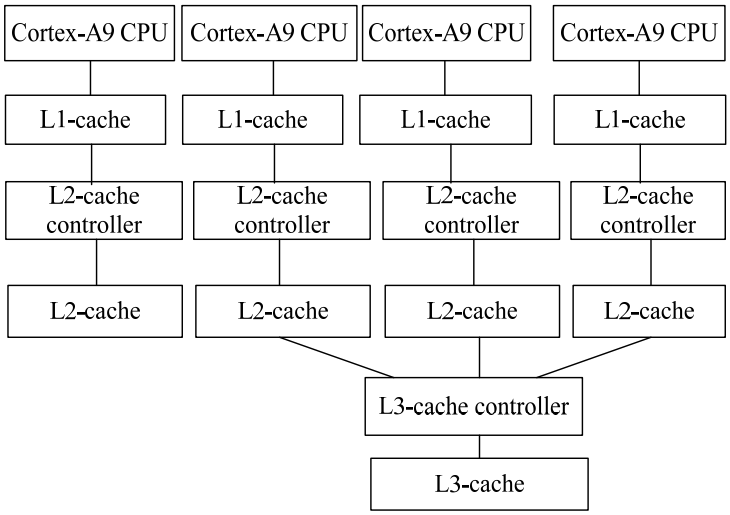

Figure 49: Diagram showing the quad-core processor model used in the performance platform model.

For performance utilization of Transport and MAC layer we only model the workload of the TCP since it is a connection-oriented, complex and more computationally intensive transport protocol then UDP. UDP is a light weight non-connection oriented transport protocol used primarily for real-time streaming applications; therefore processing costs of UDP are lower than TCP.

The workload for TCP *OS_Service* was extracted as explained in (Khan et al., 2011b). The processing times of TCP functions were scaled down since ARM-Cortex-A9 processors operate at a much faster clock then DECstation 500/200. The approximate number of abstract instructions (Khan et al. , 2011b) for the TCP OS_Service processing workload were extracted and modelled as a function workload model (Khan et al. , 2011b). This function workload model mimics the execution workload of the TCP transport protocol. It executes inside a Process workload model triggered by the TCP *OS_Service* during the processing of the service request. The Application models were abstracted out by using constant bit rate traffic generator and constant delays to measure the performance of TCP in isolation.

The average busy time of any processor core of any network node (KP or SIB in the case of M3) involved in the experiment was less than .00001% even for the case of 100 KPs. This is the processing time for TCP. The processing time of IEEE 802.11 is merely a subset of this processing time.

The claims of the authors (Khan et al., 2011b) regarding the highly efficient implementation of TCP/IP protocol (in terms of utilization of platform components) were confirmed by using CORRINA (Khan et al., 2011b). The execution times and total number of machine instructions taken by its API functions during their execution were extracted by using CORRINA (Khan et al., 2012a). The case study consists of the OfficeVideo and OfficeVideoStreamer client and server programmed using the OPENcv library as described in (Khan et al., 2011d). The execution times, total number of cache accesses and executed instructions taken by its API functions during their execution are shown in Table 7. A total of 500 frames were streamed from client to server. The client and the server applications were hosted by Intel Centrino VPro Processor based hardware platform running Ubuntu 9.04 LTS operating system.

Table 7: Execution times and run-time performance statistics of BSD API functions on Intel Centrino VPro Processor based platform running Ubuntu 9.04 LTS.

TCP/IP API function	Average Execution time	Cache accesses	executed instructions
socket	2.186 milli sec	3205	1175
bind	.171 milli sec	44525	11669
listen	.356 milli sec	40583	9236
accept	.0256 milli sec	382	119

In case of some distributed embedded systems domains such as WSNs, highly efficient MAC protocols for example nanoMAC (Haapola , 2003) and protocol stacks for example of NanoStack (Sensinode) have been implemented. Their source code can be used by the protocol designers to extract the highly accurate workload models using ABSINTH-2 for investigating their performance on a variety of platforms. Their performance can be thoroughly investigated using CORRINA on a set of intended platform families for example a variety of AMR-based platforms.

After conducting a thorough literature survey and case studies, we observed that the workload of the system calls was negligible as compared to the overall load of the application. Even in case of very simple and low complexity applications such as those comprising of simple video streamers, the load and the execution times of Transport API was a very small fraction of the overall load of the application (Khan et al., 2011d). The following conclusions related to workload extraction and accuracy can be drawn from the case studies and literature review.

1. The MAC and transport layer models are functionally accurate and follow the same trend as the corresponding ns-2 and OMNeT++ models.

2. The workload models of the highly efficient and specialized MAC protocols can be obtained via ABSINTH-2 and CORRINA.

3. The main contribution of MAC and transport protocols as far as non-functional properties are concerned is in end-to-end delays etc., but not platform utilization.

The following points can be taken into consideration while comparing the ABSOLUT MAC and Transport models to ns-2, OMNeT++ or other network simulators.

1. The functionality related to the Network layer has been abstracted out in ABSOLUT currently.

2. The alignment of traffic generators in time at the start of simulation.

3. The way system calls are modelled .i.e., blocking or non-blocking.

4. Random number generators used for back-time calculation and the seeds used for randomization can vary the simulation results significantly from other simulators

though the trend will be the same (P. Pablo et al. , 2008) and hence the models provided by these simulators are always functionally correct.

5. The queue sizes used for implementing the OS_Services must match those used in the simulators.

6. The way a collision is determined and defined in a network simulator is also very important to consider. In many network simulators, the two or more nodes will collide if they transmit at exactly the same time. In real world though, the propagation time due to distance between the nodes might also be a potential cause of collision. For example, two nodes can sense the channel idle, one starts to transmit and the other node(s) can still sense the channel idle and transmit causing collision.

The results are accurate enough for SLPE of distributed systems. The functionally correct models of System Calls will be provided as a part of ABSOLUT component library**.** Specialized probes and use cases will be provided in order to facilitate the system designer to modify and enhance the MAC and transport protocol models and make necessary adjustments as per use-case.

9.5 Accuracy of Application Workload models

As shown in in Figure 50, from an implementation perspective, an application can consist (possibly) of user-space code, external libraries and system calls. Therefore, the estimation of accuracy of corresponding application Workload models of these software implementations (user-space code, external libraries and system calls) demands the investigation of workload modelling tools employed by ABSOLUT for automatic workload generation as shown in Figure 50.

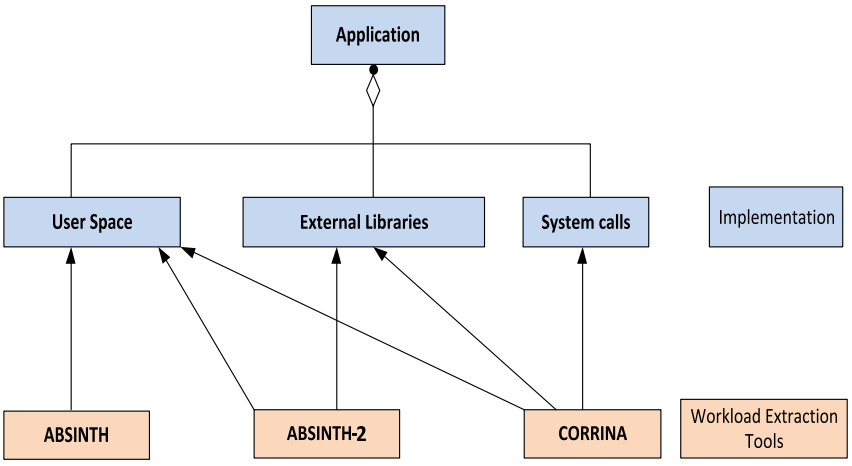

Figure 50: Investigating the accuracy of Application Workload model generating tools.

We next investigate the accuracy of workload models corresponding to User Space code and External Libraries. The accuracy of System Calls is already described in Chapter 9.4.

9.5.1 Accuracy of User Space code Workload Models

The workload models of User-Space code can be obtained by both compiler-based workload modelling tools .i.e., ABSINTH and ABSIINTH-2 and Run-Time statistics based workload modelling method CORRINA. Both ABSINTH and ABSINTH-2 employ the modified gcc compiler for workload extraction of user space code. ABSINTH is only limited to user-space code and will only generate stubs for the System Calls and External library functions. The simulation results of both the ABSINTH and ABSNITH-2 were compared to real systems in a number of case studies. A number of case studies were conducted by the principal author of the research contribution to evaluate the accuracy of ABSOLUT (Kreku et al., 2008b). The results of the case studies are shown in

Table **8**. The simulation results achieved an average accuracy of 88% and standard deviation of 12%.

Table 8: Error percentages of simulation results in case studies after comparison with real systems.

Measurement	Error
Mobile Phone Use case 1 MCU Load	25%
Mobile Phone Use case 2 MCU Load	19%
Mobile Phone Use case 2 DSP Load	13%
MP3 Playback DSP Load	0%
All-MCU MPEG4 Encoding FPS	11%
DSP-accelerated MPEG4 Encoding FPS	24%
DSP-accelerated MPEG4 Encoding DSP Load	2%
VNC use case CPU Load	15%
VNC use case FPS	11%
VNC use case network traffic	14%
Sensing algorithm execution time	3%
Average Error	**12%**
Standard Deviation	**8%**

The workload models of user-space code can be modelled manually and via run-time performance statistics based method (CORRINA). The accuracy of the manually modelled workload models depends largely on the experience of the system designer. The accuracy of the user-space workload models generated by CORRINA depends on how closely the ABSOLUT platform model used in the simulation resembles the platform on which the application was executed. The accuracy of CORRINA is greatly enhanced via the application of PAPI (Khan et al., 2012a) which covers a variety of ARM and INTEL based platforms. PAPI extracts the actual number of instructions executed by the processors in the platform for the execution of a certain lines of application source code. We therefore recommend the use of ABSINTH and ABSINTH-2 for the extraction of workload models of user-space code.

Both CORRINA and ABSINTH and ABSINTH-2 have their own advantages and disadvantages. Both ABSINTH and ABSINTH-2 are limited to gcc compiler and cannot generate workload models if a g++ compiler is used. CORRINA has no such restriction but its accuracy is dependent on the similarity of the platforms.

ABSINTH-2 is more accurate than ABSINTH in many cases where the application source code contains a lot of decisions and control. The reason is that ABSINTH uses branch probabilities generated after the application execution for decisions regarding the order of execution of function workload models etc., during the execution of process-level application workload models. On the other hand the ABSINTH-2 uses the actual application execution trace for deciding the order of execution of function workloads etc., during the execution of process-level application workload models. CORRINA also uses the actual application execution trace just like ABSINTH-2 for deciding the order of execution of function workload models in process-level application workload models. In some cases (depending on the application), the improvements resulted by using ABSINTH-2 could be quite significant (Saastamoinen, 2011b).

9.5.2 Accuracy of External Libraries workload models

ABSINTH is a compiler based technique which cannot generate the workload models for external libraries. In order to generate workload models of external library workload models, ABSINTH-2 employs SAKE tool (Saastamoinen, 2011b). SAKE tool is very advantageous since it not only generated the workload models of external libraries via Valgrind but also improves the control of the execution of the workload models by closely mimicking the order of execution of functions during the actual application simulation (Saastamoinen, 2011b). It has been proved via a case study in (Saastamoinen , 2011b) that applying SAKE tool instead of manual workload modelling of external libraries function workloads improves the simulation accuracy significantly in case of applications where the external libraries functions contribution to most of the load during the application execution. The case study conducted in (Saastamoinen, 2011b) reveals that the workload models taking external library functions were most accurate. All the estimated ABSOLUT abstract instructions such as the read, write and execute were optimistic, for example the deviation of execute instructions was limited to merely +/-25% (Saastamoinen, 2011b).

9.6 Middleware and Higher Level Protocols

Both the middleware technologies such as NoTA SOA and the higher level protocols such as SSAP in M3 (Lappeteläinen, Antti et. al., 2008) are mostly available as user space code which could be compiled as a library or readymade library. In both the cases, ABSINTH-2 can be used to extract highly accurate workload models as described in Chapter 9.5. During the research presented in the thesis, we focused on with one middleware technology .i.e., NoTA (Lappeteläinen, Antti et. al., 2008)**.** We also focused on the impact on performance simulation of the SSAP protocol employed at the information level (Eteläperä , 2011) in M3. Once the middleware and higher level protocols are developed, CORRINA (Khan et al., 2012a) can be used by the designers and developers to test the feasibility in terms of the developed solutions in real time on a number of target devices before their deployment. CORRINA gathers the run-time performance statistics during the application execution and can generate the workload models for SLPE but when the models are mapped to a variety of different ABSOLUT platform models, the dissimilarities in platforms can lower the accuracy of results. It is a very convenient tool to test the developed middleware technologies, Transport and other protocols in real devices and use cases. Usually, the middleware technologies are implemented as user space code or external libraries and their workload models can be obtained by using ABSINTH-2.

A complete case study was conducted and the contribution of NoTA device interconnect protocol (DIP) was evaluated in proportion to the overall workload of the application. NoTA DIP was employed in both modes, .i.e., compiled as a library and linked to the application (single process mode) and as a background process (Khan et al., 2010c). NoTA consists of two layers .i.e., The DIP defines both socket based communication i.e., it supports both message and streaming type of data flows. NoTA DIP is divided into two main functional blocks. The first one is called high interconnect (H_IN) which manages service registration, discovery, access and security. The second is called low interconnect (L_IN) which is responsible for connecting the subsystems together. We conducted a number of case studies to evaluate the performance of NoTA DIP and SSAP.

The first case study application consists of a very lightweight application which consists of a Test_AN and Test_SN which are video streaming client and server respectively as described in (Khan et al., 2010c). The performance results of both NoTA H_IN and L_IN when both the network nodes use NoTA DIP in SP mode are shown in Figure 51. The average contribution of NoTA DIP (both H_IN and L_IN) is well below .1% during the use-case which involves streaming of 500 packets from server to client node .i.e., AN and SN. This shows that NoTA is highly efficient since the application is extremely lightweight. The performance results are computed using binary instrumentation via Valgrind tool (Khan et al., 2010c).

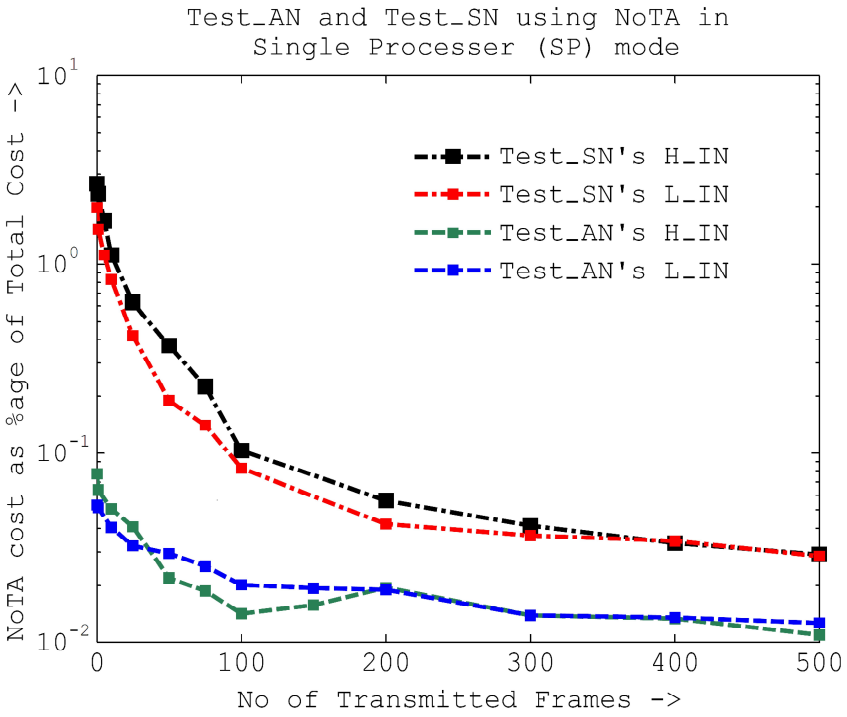

Figure 51: Performance results of NoTA DIP when operating in SP mode.

In the daemon mode, NoTA run as a daemon in the back ground and serve the ANs and SNs. The average overheads of L_IN were below .000001%; therefore the profiling results of only H_IN are shown in Figure 12. The average overheads of NoTA daemon H_IN are also below .01%. The use case consists of streaming of 1000

packets from Test_AN to Test_SN. The functionality and implementation of both the nodes remains the same as in the previous test.

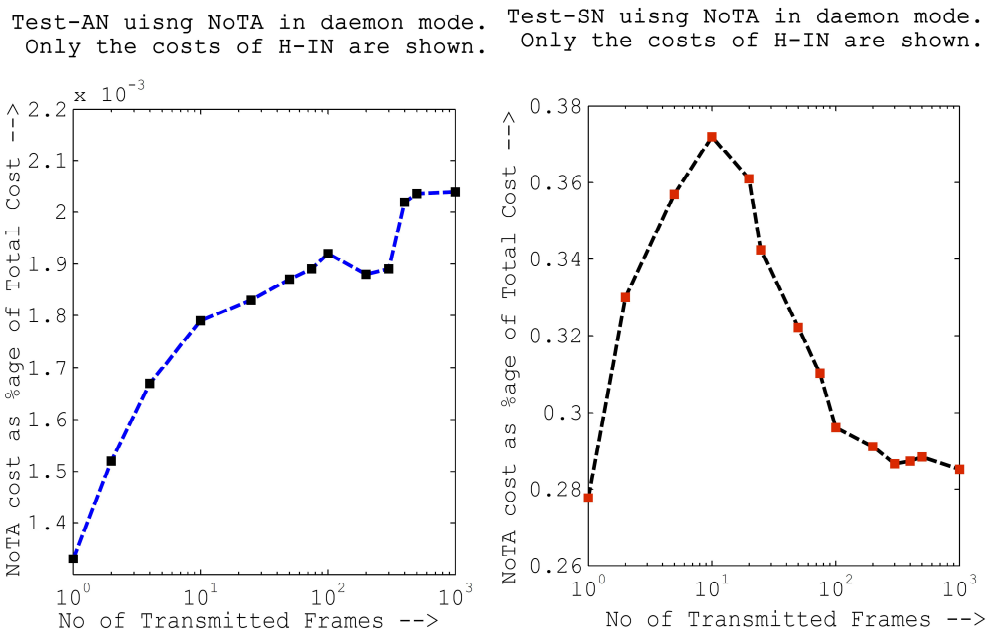

Figure 45: Performance results of NoTA DIP when operating in Daemon mode.

In order to validate the results obtained via binary instrumentation, the use-case was repeated with the both the nodes .i.e., Test_AN and Test_SN using NoTA in SP mode. Instead of Valgrind, CORRINA (Khan et al., 2012a) was used to evaluate the performance of NoTA DIP. The execution times of the NoTA API functions clearly indicate the lightweight implementation of the NoTA DIP. The proportion of (sum of) execution times of the NoTA DIP functions as compared to the overall execution time of the application was found less than 1% which validates the results obtained by binary instrumentation. The results are shown in Table 9.

Table 9: The processing times of NoTA API functions obtained via CORRINA.

NoTA API functions	Average Execution Times of NoTA API functions on Intel Core i5 Processor based platform
Hsocket	3196 usec
Hbind	446 usec
Hlisten	374 usec
Haccept	592 usec
Hrecvive	442 usec
Hsend	429 usec

Therefore from the results obtained via binary instrumentation and CORRINA indicate that NoTA does not act as a potential bottleneck even for very lightweight distributed applications. The implementation of the Test_AN and Test _SN is described in (Khan et al., 2010c)**.**

Next, we evaluate the performance of a higher level protocol .i.e., SSAP API functions. In M3, SSAP API is used by KPs and an M3 information level interoperability solutions for example RIBS (Eteläperä, 2011) and SIB (Lappeteläinen, Antti et. al., 2008). The processing Times of the SSAP API functions on an Intel Core i5 Processor based platform are shown in Table 10. As a consequence, the processing times of the SSAP API functions are less as compared to NOTA API functions shown in Table 9.The processing times were measured via CORRINA (Khan et al., 2012a). The results shown in Table 10 were obtained by using SSAP used over NOTA (at service level) with TCP/IP (at transport level). After analysing the processing time of SSAP API functions, it was observed that the implementation of SSAP API is even more light weight than the service level IOP solutions .i.e., NoTA and ADIOS. The test bench shown in Figure 52 was used for performance evaluation of M3. The Perf_KP sends different types of SSAP messages to RIBS via SSAP and the execution times and run-time statistics of these functions are recorded.

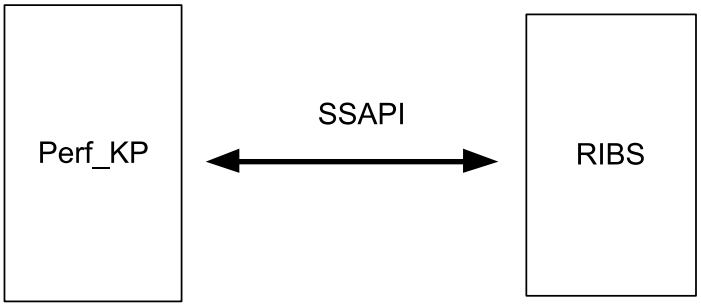

Figure 52: Test bench for recording SSAP delays and performance statistics.

The Average execution times of the SSAPI API functions were recorded using CORRINA and are shown in Table 10.SSAP API functions execution times contribute to a negligible fraction of the total execution times of the KPS and RIBS.

Table 10: SSAP delays obtained via the test bench (Perf_KP and RIBS pair).

SSAPI API functions	Average Execution Times on Intel Core i5 Processor based platform
SsInsert	49 usec
SsQuery	291 usec
SsUpdate	53 usec
SsSubscribe	58 usec
SsRemove	47 usec
SsJoin	68 usec
SsLeave	57 usec

We therefore conclude that SSAP is a very light weight protocol and the performance overheads and delays resulting from using it as a means of information exchange between RIBS and KPs are negligible. From the aforementioned facts, it can be concluded that the middleware technologies and other higher level protocols considered .i.e., NoTA and SSAP were very lightweight and available as external libraries or open source code. In both the cases highly accurate workload models can be obtained via ABSINTH-2 in the same way as explained in Chapter 9.5.2. In rare cases where the middleware is implemented as services available via the operating system, the workload models can either be generated via CORRINA or manually modified to reflect the differences in the platforms. The accuracy of manual models will of course depend on the experience of the system designer. The accuracy of CORRINA depends on the differences between the ABSOLUT platform models and the platforms used to run the application for extracting workload models. It was observed that the workload of middleware and other protocols is very low as compared to the overall workload of the application. Even for very lightweight applications we found it to be less than 1%. Also their role in non-functional properties such as end-to-end delays can be neglected since the processing times of API functions of middleware and higher level protocols were less them 1 milli second in the case studies presented in research articles. Therefore, the design decisions taken by the system designer during architectural exploration are not affected by their workload values.

9.7 Summary

Highly accurate workload models for system calls, external libraries and user-space code can be created and integrated to ABSOLUT. ABSOLUT provides a high level of automation for the workload extraction via tools such as ABSINTH. ABSINTH-2 and CORRINA. The accuracy of the workload models has been presented in a number of case studies and research articles as described in Chapter 9.3, Chapter 9.4 and Chapter 9.5.

10. Introduction to papers

Paper I (Linking GENESYS application architecture modelling with platform performance simulation. FDL2009) describes the solutions to two research problems, .i.e., how the application architecture design methodology can be linked to the performance simulation phase? and how the non-functional properties of the system can be validated in the performance simulation phase? It should be noted that this paper applies the proposed solution in a case study consisting of a non-distributed (all application components residing on the same device) application which was not the ultimate goal of the research. Nevertheless, the generic approach described in this paper can be applied to distributed systems without any modification. While this research was being conducted, the Transport, Data-link and Middleware models were not modelled/integrated in ABSOLUT. The next logical step was to test the approach of back-to-back application design and performance simulation by employing highly abstract communication models between processes in the ABSOLUT application models. This means that the Transport and Data-link layers will be abstracted out by employing communication mechanisms employed by well-known MOCs such KPN MOC.

Paper II (Performance evaluation of distributed embedded systems applications via Kahn process networks and ABSOLUT. UBICOMM 2011) focuses on the performance evaluation of distributed applications via ABSOLUT via abstracting the Transport and Data-link layers of the OSI model. At first, the paper elaborates the properties of the KPN MOC that must be satisfied by its instantiation if the affective adaption of the KPN MOC is desired. Afterwards, the paper describes the was KPN MOC was affectively adapted over ABSOLUT and the way the KPN MOC concepts correlate with the ABSOLUT application workload modelling concepts/models. Afterwards, the concepts gathered in Paper V are applied to integrate the application modelling phase and workload modelling phase for a distributed application. This confirms the fact that the seamless integration of application design and performance simulation achieved in Paper IV is not limited to non-distributed applications. Also the non-functional properties can be easily validated via the performance simulation results in the same was as non-distributed applications. The contribution of this article is concisely presented in Chapter 6.

Paper III (Multi-threading support for system-level performance simulation of multi-core architectures. ARCS 2011) discusses the way multithreading support is provided in ABSOLUT. The approach enables the modelling of ABSOLUT workload models for multithreading applications. The approach is experimented with a case study described in this research article. The modelling of methodology is presented in Chapter 4 of the thesis. The inter-process communication and system-synchronization mechanisms enable the communication between ABSOLUT process models which closely mimic the real world IPC models. The main contribution in this research article was made by Jukka Saastamoinen who is the first author of this research article.

Paper IV (Analysing transport and MAC layer in system-level performance simulation. SoC 2011) elaborates the design and integration of functional MAC and Transport Models to ABSOLUT framework. The paper describes the modelling of UDP transport protocol and IEEE 802.11 DCF contention resolution data-link layer protocols. The paper also illustrates the performance evaluation of these protocols in isolation via traffic generators. The models described in this article are used to evalu-

ate the performance of distributed embedded systems where these protocols play a key role in end-user experience for example end-to-end packet delays and packet loss rate etc., . The models described in this article are explained in Chapter 5.1, 5.2, 5.3 and 5.4. The results obtained by employing these models are compared with ns-2 and OMNeT++ network simulators and confirm the functional correctness of the modelled protocols since the results show the similar trends and absolute values under the same scenarios and network configuration.

Papers V (Application Workload Modelling via Run-Time Performance Statistics. IJERTCS 2012) focuses on the description of a novel non-compiler based methodology for automatic workload generation for ABSOLUT called CORRINA. The methodology records the run-time statistics during the execution of the application by accessing the hardware performance counters of platforms via PAPI (Papi). When the application execution terminates, it utilizes these statistics for generating the ABSOLUT function workload models. The methodology also generates the trace record of the order in which the different functions corresponding to the function workload models were triggered during the application execution. This trace information is used by the ABSOLUT process workload models to trigger the function workload models in the same sequence as during the execution of the application. The methodology also generates the function workload models for middleware technologies if they are available as user space open source software, libraries or implemented inside the OS Kernel as system calls. It also generates the workload models for any other system calls such as TCP/UDP API functions and external libraries. The methodology is described in Chapter 7.

Paper VI (SLPE of distributed GENESYS applications on multi-core platforms. EmbeddedCom 2011) and Paper VII (System-level performance evaluation of distributed multi-core NoTA systems. NESEA 2011) describe the performance evaluation of distributed GENESYS and NoTA applications via ABSOLUT. The Transport and data-link layer models described in Paper IV are used to mimic the functionality of Transport, Data-link layer protocols in real world devices. In Paper VII, the ABSOLUT models corresponding to different models of NoTA DIP .i.e., Daemon mode, Kernel Implementation and Single Process mode are also elaborated. The models, tools and techniques used in the case studies presented these articles span Chapter 5 to Chapter 8 of thesis.

11. Conclusions and discussion

Distributed embedded and computer systems are increasingly spanning a wide range of industrial domains for example automotive industry, energy sector and high-end mobile phones/network tablets. Modern cars contain a wide range of networked sensors and ECUs (electronic control units) for example a new modern Mercedes S-Class car contains over 70 networked ECUs. Just a decade back, most cars had only three ECUs. No one could imagine that small network computer systems called wireless sensor networks (WSNs) will help in reducing the energy consumption of the traffic lights. Also, a wide range of functionalities supported/implemented by desktop computers of a few years back have converged to the high-end mobile phones of today. No one could play a High definition music video on a mobile phone a decade back. The smartest phones of a decade back have less functionalities/features then many low-end basic mobile phones of today. The stiff competition in the mobile phone market segment and the way innovations are being made to make the end-user experience more enjoyable is mind blowing. All this has been possible due to the advancements in hardware, software and networked technologies which have played a dramatic role in reducing the form factors and increasing the computational capacity of the devices many folds simultaneously.

The complexity of the networked computer systems is increasing due to increased functionalities .The technologies employed by these systems such as Data-link layer protocols, Transport Layer protocols and Middleware technologies are evolving at a rapid pace in order to satisfy the non-functional properties of these systems. This is precisely the reason for the usage of specialized data-link protocols such as MAC contention resolution schemes for WSNs. Likewise, in a wide variety of distributed multimedia applications, middleware technologies such are used to aid the applications by providing different functionalities in the form of API functions. This also aids the application developer to focus on the functional and implementation aspects of the application which provide the end-user with unique experience. This allows product differentiation with minimal effort.

In the beginning, the communication systems were mostly point to point solutions like letters and telephones or broadcast solutions (one-to-many) for example radio and the TV. The modern communication solutions for example computers, the Internet and mobile phones provide users with a possibility to get access to vast amount of information and services. These systems are still heavily dependent on human guidance. Now a days, the trend is to look for new solutions which will make the information available to everywhere in a physical space without human assistance. Such physical spaces consisting of many heterogeneous embedded devices and services which can interact automatically for exchange information on behalf of an individual user are often called Smart Spaces or Smart Environments. The interoperability challenges within Smart Environment interoperability must be resolved in order to enable reliable information sharing and communication between devices in a smart environment in a meaningful way.

Many vendor specific interoperability solutions have been provided for information sharing in a smart environment. For example Samsung manufactures high-end mobile phones as well as household appliances such as Televisions, Washing machines, Microwave ovens and refrigerators etc., and has provided applications for its high-end mobile phones which communicate with other household appliances to retrieve information which can be used to control these devices manually and remotely. The appliances can inform the mobile phone application about their status after which the user can control

them by sending different commands. The M3 is a generic solution which is not vendor specific. Even in the vendor specific solutions, the interacting devices must overcome the interoperability challenges. Thus the devices must contain software components that will resolve the interoperability issues among devices in a smart environment at the levels explicitly mentioned in M3. Such solutions are expected to be much simpler and optimized since the vendor usually allocates sufficient amount of funds for the related research and development activities. Also, the devices which these solutions will span are usually known beforehand by the vendor. From the aforementioned discussion, it is concluded that a number of different technologies have been currently employed in the distributed networked computer and embedded systems depending on the application domain.

Therefore, the complexity of these systems is growing rapidly. In order to achieve the faster deployment and more optimal design of these systems, their feasibility must be evaluated at an early design phase. If the source code of the application is available, the methodology must extract the workload models of the application automatically via specialized tools to evaluate its feasibility on a variety of different platforms. If the application source code of the modelled application is not available, the application model layers must act as a blue print for the identification of application workload models. In that case, the workloads corresponding to each application workload model layer can be estimated by studying the algorithmic details of the application. In that case, of course the extent to which the application models reflect the workload and control of the actual application (to be modelled) depends on the experience of the system designer in that application domain. In both the cases, the probability of a more robust and optimal system design is enhanced. In order for a methodology to be applicable across multiple domains of distributed systems and applications, the methodology must provide the easy instantiation of Transport and data-link protocol models, workload modelling of middleware and higher level protocols.

The thesis first presents the merits on the basis of which ABSOLUT was preferred over other SLPE approaches for extension to a system and application domain independent methodology for the SLPE of distributed embedded systems. Afterwards, the thesis mentions the models and tools which the ABSOLUT methodology must provide in order to be employed for the SLPE of distributed systems/applications. The rest of the thesis describes the modelling and integration of the models/tools to ABSOLUT. New models and protocols can be added to the framework as they are developed and used in the performance models for SLPE. The functionalities that are common among these protocols are implemented in the base classes so that new models can be developed with minimal effort. The modelled protocols and tools developed during this research were extensively applied in a number of case studies which demonstrate the performance modelling and evaluation of GENESYS, NoTA and M3 applications via the extended ABSOLUT framework.

Research Topics for the future: The network layer protocols of the OSI model were not modelled and the research efforts were mainly directed for the SLPE of single hop wireless systems. The network protocols can be easily instantiated by using the OS_Service base class models used for modelling the Transport and data-link layer protocols. The protocols at data-link and transport layer are functionally correct and can therefore the functional network layer protocols can be integrated into the framework just like transport and data-link Layer protocols.

Also, in order for the automatic workload generation methods to span applications other than C/C++, an LLVM compiler based workload generation tools is under development which will give the

coverage of FORTRAN based applications and potentially a wide variety of Java applications. Unlike GCC, the LLVM compiler is very well documented and is designed using object oriented paradigm which allows for easy modification.

Furthermore, the models for the estimating the energy consumption of Data-link and Transport protocols are planned to be integrated to ABSOLUT framework. Also, the models/methodologies for estimation of energy consumption due to user-space code, external libraries and system calls used by an application are under development.

References

Baghdadi A, Zergainoh N, Cesario W & Jerraya AA (2002) Combining a performance estimation methodology with a hardware/software codesign flow supporting multiprocessor systems. IEEE Transactions on software engineering 28(9): 822–831.

Baghdadi A, Zergainoh N, Cesario W, Roudier T & Jerraya AA (2000) Design space exploration for hardware/software codesign of multiprocessor systems. Proc. 11th International Workshop on Rapid System Prototyping (RSP), 8–13.

Beltrame G, Bolchini C, Fossati L, Miele A & Sciuto D (2008) Resp: A non-intrusive transactionlevel reflective mpsoc simulation platform for design space exploration. Proc. Proceedings of the 2008 Asia and South Pacific Design Automation Conference.

Bobrek A, Pieper J, Nelson J, Paul J & Thomas D (2004) Modeling shared resource contention using a hybrid simulation/analytical approach. Proc. Proceedings of the Design, Automation and Test in Europe, 1144–1149.

E. Pitoura and B. Bhargava, “Data consistency in intermettently connected distributed systems,” IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No. 6, November 1999, pp. 896–915.

Eteläperä, Matti; Keinänen, Kari; Kiljander, Jussi. Feasibility Evaluation of M3 Smart Space Broker Implementations Applications and the Internet (SAINT), 2011 IEEE/IPSJ 11th International Symposium on (2011), 292-296

Fornaciari W, Sciuto D, Silvano C & Zaccaria V (2001) A design framework to efficiently explore energy-delay tradeoffs. Proc. Ninth International Symposium on Hardware/Software Codesign (CODES), 260–265.

Fornaciari W, Sciuto D, Silvano C & Zaccaria V (2002) A sensitivity-based design space exploration methodology for embedded systems. Design Automation for Embedded Systems 7(1–2).

G. Forman and J. Zahorjan, “The challenges of mobile computing,” IEEE Computer, Vol. 27, No. 4, April 1994, pp. 38–47.

Insup Lee (Editor), Joseph Y-T. Leung (Editor), Sang H. Son. Handbook of Real-Time and Embedded Systems. Publisher: Chapman and Hall/CRC (July 23, 2007) .800 pages. Language: English.ISBN-10: 1584886781. ISBN-13: 978-1584886785.

J. Haapola, "NanoMAC: A Distributed MAC Protocol for Wireless Ad Hoc Sensor Networks," Proc. XXVIII Convention on Radio Science & IV Finnish Wireless Communication Workshop, pp. 17-20, 2003.

J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, System Architecture Directions for Networked Sensors, ASPLOS, November 2000.

J. Polastre, J. Hill, and D. Culler, Versatile Low Power Media Access for Wireless Sensor Networks, ACM SenSys, November 2004.

Jaber C, Kanstein A, Apvrille L, Baghdadi A, Moenner PL & Pacalet R (2009) High-level system modeling for rapid hw/sw architecture exploration. Proc. IEEE/IFIP International Symposium on Rapid System Prototyping (RSP ’09), Paris, France, 88–94.

K. Tachikawa, “A perspective on the evolution of mobile communications,” IEEE Communications Magazine,Vol. 41, No. 10, October 2003, pp. 66–73.

Kangas T, Kukkala P & Orsila H (2006) Uml-based multiprocessor soc design framework. ACM Transactions on Embedded Computing Systems 5(2): 281–320.

Khan, Subayal; Ovaska, Eila; Tiensyrjä, Kari; Nurmi, J.(2010a). From Y-chart to seamless integration of application design and performance simulation .Proceedings 2010 International Symposium on System-on-Chip - SOC. Tampere, Finland, 29-30 Sept. 2010. IEEE. Piscataway, NJ, USA (2010), 18-25

Khan, Subayal; Pantsar-Syväniemi, Susanna; Kreku, Jari; Tiensyrjä, Kari; Soininen, Juha-Pekka. Linking GENESYS application architecture modelling with platform performance simulation. Forum on Specification and Design Languages 2009 (FDL2009). Sophia Antipolis, France, September 22-24, 2009. ECSI. France (2009)

Khan, Subayal; Saastamoinen, Jukka; Huusko, Jyrki; Nurmi,Jari. (2011a).Performance evaluation of distributed embedded systems applications via Kahn process networks and ABSOLUT. The Fifth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies UBICOMM 2011. Lisbon, Nov. 20-25, 2011. IARIA (2011)

Khan, Subayal; Saastamoinen, Jukka; Majanen, Mikko; Huusko, Jyrki; Nurmi, Jari. (2011b). Analyzing transport and MAC layer in system-level performance simulation. 2011 International Symposium on System on Chip, SoC 2011. Tampere, Finland, 31 Oct. - 2 Nov. 2011. IEEE Computer Society (2011), 8 p.

Khan, Subayal; Saastamoinen, Jukka; Nurmi,Jari. (2011c).System-level performance evaluation of distributed multi-core NoTA systems. 2nd IEEE International Conference on Networked Embedded Systems for Enterprise Applications. NESEA 2011, Fremantle, Dec. 8-9, 2011. IEEE (2011)

Khan, Subayal; Saastamoinen,Jukka; Tiensyrjä, Kari; Nurmi, Jari. (2011d).SLPE of distributed GENESYS applications on multi-core platforms. The 9th IEEE international symposium on Embedded Computing (EmbeddedCom 2011). Sydney, Dec 12-14 2011

Khan, Subayal; Tiensyrjä, Kari; Nurmi, Jari. (2010b).Instantiating GENESYS application architecture modelling via UML 2.0 constructs and MARTE profile. Proceedings - 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools, DSD 2010. Lille, France, 1-3 Sept. 2010. EUROMICRO. France (2010), 251-254

Khan, Subayal; Tiensyrjä, Kari; Nurmi, Jari. (2010c). Instantiation and feasibility evaluation of NoTA SOAD via MARTE profile and binary instrumentation.The 7th International Conference and Expo on Emerging Technologies for a Smarter World. CEWIT 2010. Incheon, 27 - 29 Sep. 2010

Khan, Subayal;Saastamoinen, Jukka;Jyrki,Huusko; Nurmi, Jari. (2012a). “Application Workload Modelling via Run-Time Performance Statistics”. International Journal of Embedded and Real-Time Communication Systems (IJERTCS), 2012.

Kienhuis, E. Deprettere, K. Vissers and P. van der Wolf. Approach for quantitative analysis of application-specific dataflow architectures,” in Proceedings of the IEEE International Conference on Application- Specific Systems, Architectures and Processors (ASAP ’97), pp. 338– 349, Zurich, Switzerland. July 1997.

Kiljander, Jussi; Keinänen, Kari; Eteläperä, Matti; Takalo-Mattila, Janne; Soininen, Juha-Pekka.Autonomous file sharing for smart environments.PECCS 2011, Vilamoura, Algarve, Portugal, March 5 - 7, 2011PECCS 2011 Proceedings of the 1st International Conference on Pervasive and Embedded Computing and Communication Systems (2011), 191-196

Kreku J & Soininen J (2003) Mappability estimate: a measure of the goodness of a processoralgorithm pair. Proc. System-on-Chip, 2003. Proceedings. International Symposium on, IEEE, 119–122.

Kreku J & Tiensyrjä K (2011) Scalable Multi-core Architectures: Design Methodologies and Tools, chapter System exploration. Springer.

Kreku J, Eteläperä M & Soininen JP (2005) Exploitation of UML 2.0-based platform service model and SystemC workload simulation in MPEG-4 partitioning. Proc. International Symposium on System-on-Chip Proceedings, 167–170.

Kreku J, Hoppari M, Kestila T, Qu Y, Soininen J & Tiensyrja K (2008a) Application-platform performance modeling and evaluation. Proc. Specification, Verification and Design Languages,2008. FDL 2008. Forum on, IEEE, 43–48.

Kreku J, Hoppari M, Kestilä T, Qu Y, Soininen JP & Tiensyrjä K (2009) Languages for Embedded Systems and their Applications, volume 36 of Lecture Notes in Electrical Engineering, chapter Application Workload and SystemC Platform Modeling for Performance Evaluation, 131–148. Springer.

Kreku J, Hoppari M, Kestilä T, Qu Y, Soininen JP, Andersson P & Tiensyrjä K (2008b) Combining uml2 application and systemc platform modelling for performance evaluation of real-time embedded systems. EURASIP Journal on Embedded Systems .

Kreku J, Hoppari M, Tiensyrjä K & Andersson P (2007) Systemc workload model generation from uml for performance simulation. Proc. Forum on Specification and Design Languages.

Kreku J, Kauppi T & Soininen JP (2004a) Evaluation of platform architecture performance using abstract instruction-level workload models. Proc. International Symposium on System-on-Chip Proceedings, 43–48.

Kreku J, Penttilä J, Kangas J & Soininen JP (2004b) Workload simulation method for evaluation of application feasibility in a mobile multiprocessor platform. Proc. Proceedings of the Euromicro Symposium on Digital System Design, 532–539.

Kreku J, Qu Y, Soininen JP & Tiensyrjä K (2006) Layered uml workload and systemc platform models for performance simulation. Proc. International Forum on Specification and Design Languages (FDL), 223–228.

Kreku J, Tiensyrjä K & Vanmeerbeeck G (2010) Automatic workload generation for system-level exploration based on modified gcc compiler. Proc. Design, Automation and Test in Europe conference and exhibition.

Lahiri K, Dey S & Ragunathan A (2001a) Evaluation of the traffic-performance characteristics of system-on-chip communication architectures. Proc. Proceedings of 14th International Conference on VLSI Design, 29–35.

Lahiri K, Ragunathan A & Dey S (2000a) Efficient exploration of the soc communication architecture design space. Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD), 424–430.

Lahiri K, Ragunathan A & Dey S (2000b) Performance analysis of systems with multi-channel communication architectures. Proc. Proceedings of the 13th International Conference on VLSI Design, Calcutta, India, 530–537.

Lahiri K, Ragunathan A & Dey S (2001b) System-level performance analysis for designing on-chip communication architectures. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 20(6): 768–783.

Lahiri K, Ragunathan A & Dey S (2004) Design space exploration for optimizing on-chip communication architectures. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 23(6): 952–961.

Lappeteläinen, Antti et. al., 'Networked Systems, Services, and Information - The Ultimate Digital Convergence'. 1st International Conference on Network on Terminal Architecture, June 11, 2008, Helsinki.

Lieverse P, van der Wolf P & Deprettere E (2001a) A trace transformation technique for communication refinement. Proc. 9th International Symposium on Hardware/Software Codesign (CODES), 134–139.

Lieverse P, van der Wolf P & Deprettere E (2001a) A trace transformation technique for communication refinement. Proc. 9th International Symposium on Hardware/Software Codesign (CODES), 134–139.

Lieverse P, van der Wolf P, Vissers K & Deprettere E (2001b) A methodology for architecture exploration of heteregeneous signal processing systems. Kluwer Journal of VLSI Signal Processing 29(3): 197–207.

Lieverse P, van der Wolf P, Vissers K & Deprettere E (2001b) A methodology for architecture exploration of heteregeneous signal processing systems. Kluwer Journal of VLSI Signal Processing 29(3): 197–207.

M.Weiser, “Some computer science issues in ubibuitous computing,” Communications of the ACM, Vol. 36, No. 7, July 1993, pp. 75–84.

Mahadevan S, Angiolini F, Storgaard M, Olsen R, Sparso J & Madsen J (2005a) A network traffic generator model for fast network-on-chip simulation. Proc. Proceedings of the Design, Automation and Test in Europe, 780–785.

Mahadevan S, Storgaard M, Madsen J & Virk K (2005b) Arts: a system-level framework for modeling mpsoc components and analysis of their causality. Proc. Proceedings of the International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 480–483.

Mohanty S & Prasanna V (2002) Rapid system-level performance evaluation and optimization for application mapping onto soc architectures. Proc. Proceedings of the IEEE International ASIC/SOC Conference, 160–167.

Mohanty S, Prasanna V, Neema S & Davis J (2002) Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. ACM SIGPLAN Notices 37(7): 18–27.

Moore,S., Ralph, J. "User-defined Events for Hardware Performance Monitoring," ICCS 2011 Workshop: Tools for Program Development and Analysis in Computational Science, Singapore, June 1, 2011.

N. Shah. Understanding network processors. Master's thesis, Dept. of Electrical Eng. and Computer Sciences, University of California, Berkeley, September 2001.

P. Pablo Garrido, Manuel P.Malumbres and Carlos T.Calafate. ns-2 vs. OPNET: a comparative study of the IEEE 802.11e technology on MANET environments. SIMUTOOLS 2008 - 1st International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems.

P. Pablo Garrido, Manuel P.Malumbres and Carlos T.Calafate.Patryk Zadarnowski, The design and implementation of an extendible instruction set simulator,BE Thesis, School of Computer Science and Engineering, University of NSW, Sydney 2052, Australia, 2000.

Papi, accessed on 15:09:2012, 05:09 PM, URL: http://icl.cs.utk.edu/papi/

Paul J, Bobrek A, Nelson J, Pieper J & Thomas D (2003) Schedulers as model-based design elements in programmable heterogeneous multiprocessors .

Paul JM, Thomas DE & Cassidy AS (2005) High-level modeling and simulation of single-chip programmable heterogeneous multiprocessors. ACM Transactions on Design Automation of Electronic Systems 10(3): 431–461.

Paulin P, Pilkington C & Bensoudane E (2002) Stepnp: A system-level exploration platform for network processors. Design & Test of Computers, IEEE 19(6): 17–26.

Pimentel A, Hertzberger L, Lieverse P, van derWolf P & Deprettere E (2001) Exploring embedded systems architectures with artemis. IEEE Computer 34(11): 57–63.

Posadas H, Herrera F, Sanchez P, Villar E & Blasco F (2004) System-level performance analysis in systemc. Proc. Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE 2004), Paris, France, 378–383.

Saastamoinen Jukka; Khan, Subayal; Tiensyrjä, Kari; Taipale, Tapio. (2011a). Multi-threading support for system-level performance simulation of multi-core architectures. ARCS 2011. 24th International Conference on Architecture of Computing Systems 2011, Workshop Proceedings. VDE Verlag Gmbh (2011), 169-177

Saastamoinen, Jukka; Kreku, Jari. (2011b). Application workload model generation methodologies for system-level design exploration. Proceedings of the 2011 Conference on Design and Architectures for Signal and Image Processing, DASIP 2011. Tampere, Finland, 2-4 Nov. 2011. IEEE Computer Society (2011), 254-260

Sensinode. Website: http://www.sensinode.com/

Simon Perathoner, Ernesto Wandeler, Lothar Thiele, Arne Hamann, Simon Schliecker, Rafik Henia, Razvan Racu, Rolf Ernst, Michael González Harbour: Influence of Different Abstractions on the Performance Analysis of Distributed Hard Real-Time Systems. Design Automation for Embedded Systems, Springer Science+Business Media, LLC, accepted 2008.

Steve Vinoski . CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, IEEE Communications Magazine, Vol. 14, No. 2, February, 1997.

Terpstra F, Polstra S, Pimentel A & Hertzberger B (2001) Rapid evaluation of instantiations of embedded systems architectures: A case study. Proc. In Proc. of the Progress workshop on Embedded Systems, Citeseer.

Vahdat, A. Lebeck, and C. S. Ellis, “Every Joule is precious: The case for revisiting OS design for evergy efficiency,” Proceedings of the 9th ACM SIGOPS European Workshop, September 2000, pp. 31–36.

Valgrind, accessed on 16:09:2012, 05:04 AM, URL: http://valgrind.org/

Weiwei Chen, Rainer Dömer, "ConcurrenC: A new approach towards effective abstraction of C-based SLDLs", in Proceedings of the International Embedded Systems Symposium, "Analysis, Architectures and Modeling of Embedded Systems" (ed. A. Rettberg, M. Zanella, M. Amann, M. Keckeisen, F. Rammig), Springer, Langenargen, Germany, September 2009.

Wild T, Herkersdorf A & Lee GY (2006) Tapes—trace-based architecture performance evaluation with systemc. Design Automation for Embedded Systems 10(2–3): 157–179. Special Issue on SystemC-based System Modeling, Verification and Synthesis.

Zivkovic V, de Kock EA, van der Wolf P & Deprettere E (2003a) Fast and accurate multiprocessor architecture exploration with symbolic programs. Proc. Proceedings of the conference on Design, Automation and Test in Europe.

Zivkovic V, Deprettere E, van der Wolf P & de Kock EA (2002) Design space exploration of streaming multiprocessor architectures. Proc. IEEE Workshop on Signal Processing Systems (SIPS), 228–234.

Zivkovic V, Deprettere V, van der Wolf P & De Kock E (2003b) From high level application specification to system-level architecture definition: Exploration, design and compilation .

PUBLICATION I

Linking GENESYS Application Architecture Modelling with Platform Performance Simulation

Subayal Khan, Susanna Pantsar-Syväniemi, Jari Kreku, Kari Tiensyrjä and Juha-Pekka Soininen

Technical Research Centre of Finland (VTT), Kaitoväylä 1, FI-90571 Oulu, Finland
Email: {subayal.khan, susanna.pantsar-syvaniemi, jari.kreku, kari.tiensyrja, juha-pekka.soininen}@vtt.fi

Abstract

Modern mobile devices support diverse applications that are enabled by rapid increase of computational power of mobile platforms. A brisk performance evaluation phase is required after the application modelling to evaluate feasibility of new applications on the platform. GENESYS modelling methodology employing service-oriented component based application design has been extended for this purpose such that application level services are refined to platform-level services allowing mapping of GENESYS application architecture to workload models used in performance evaluation. The approach is experimented with a case study. MARTE UML2.0 profile, supporting Papyrus modelling tool and SystemC were used for modelling and simulation.

1. Introduction

Mobile handheld multimedia devices are evolving at a rapid pace due to diverse audio and multi-media applications [1]. To support these computationally intense applications, heterogeneous multiprocessor architectural platforms are used. Deployment of a new multimedia application is challenging not only due to the heterogeneous parallelism in the platforms [2], but also due to performance and energy constraints. For efficient product development, it is of pivotal importance that the application design phase of software life cycle provides the foundation for system-level performance evaluation, reducing time and effort involved therein. To address such issues as design complexity, time-to-market and first time success, platform-based design was introduced a decade ago [2]. A platform is defined as an abstraction layer facilitating a set of possible refinements into a subsequent abstraction layer in the design flow [3].

In model based software development, primary artefacts of development are models [4]. A model is defined as “a reduced representation of system highlighting properties of interest from a given viewpoint". Models facilitate easier understanding of complex systems and are useful for all the phases of system life cycle. Y-chart [5] scheme is commonly applied for designing heterogeneous systems, segregating the application and architecture modelling. The application model is mapped onto platform model for analysing properties of the system model [6]. GENESYS [7] was employed for application modelling in this work. For performance evaluation, a model-based approach was adopted for both application and platform [8]. The application model was mapped to the platform model for system-level performance simulation. The main contribution of this paper is to link the GENESYS application modelling approach to the performance evaluation strategy.

Rest of paper is organized as follows: Chapter 2 describes related work. Chapter 3 explains GENESYS methodology. Chapter 4 describes performance modelling approach. Chapter 5 shows extensions to GENESYS for integration with performance evaluation strategy. Chapter 6 shows simulation results, conclusions are drawn in Chapter 7. Chapter 8 is for acknowledgements. References are listed in Chapter 9.

2. Related Work

Software architecture expresses the system in the form of different views, each representing a different aspect of the system [9]. Object Management Group (OMG) defines Model Driven Application Architecture (MDA) relying on efficient use of system models to facilitate transformations between different model types [4]. Unified Modelling Language (UML) is used to describe application structure, behaviour, and architecture [10].

Various Architectural Description Languages (ADL) have been proposed. MBASE provides integrated models for capturing the product success, process and properties [11]. Acme design language relies on a core ontology comprising of seven elements representing architectural elements [12]. Mae [13] triggers the modelling, analysis, and management of different versions of architectural artefacts supporting domain-specific extensions to capture other system properties.

Performance modelling has been approached in different ways. SPADE [14] treats applications and architectures separately via a trace-driven simulation approach. Artemis [15] extends SPADE by involving virtual processors and bounded buffers. TAPES [16] abstracts functionalities by processing latencies covering the interaction of associated sub-functions on the architecture without actually running application code. MESH [17] treats resources, software and schedulers/protocols as three abstraction levels that are modelled by software threads.

The main contribution of this paper is to link the GENESYS application modelling style [18] to the performance evaluation phase [8]. This is done via an extension of the GENESYS application modelling style to form layered application architecture. Then the layers that are compatible to the workload model layers are identified and mapped to workload layers. Thus the resulting application model can be efficiently used as a starting point for performance evaluation reducing the time and effort.

3. GENESYS Application Modelling

In GENESYS, compliance of architectural views and concepts across application domains forms basis of the cross-domain architectural style. GENESYS reference architecture template provides core and optional services to application components. Core services are fundamental to any architecture. Optional services, built on top of the core services, can be used in applications across multiple domains.

3.1. GENESYS Views

The modelling process starts by describing a set of views defined in GENESYS that are sufficient for the modelling objective. This set of GENESYS views are instantiated by using UML2.0 MARTE profile and will be illustrated in conjunction with the mobile video player case study in the following section. GENESYS use case view describes the functionality of a system at a higher abstraction level by means of use cases. The structural view defines the interface between an application and the sub-systems of the execution platform, describing the core and optional services which the different sub-systems of the underlying platform offer to an application. The syntactical view describes the syntax the servers understand in order to access their services. Sub-systems together with their interfaces (set of services) are conceived as servers that admit different messages from the application (client). The *b*ehavioural view reflects the behavioural aspects of an application and its encompassing services.

3.2. Non-Functional Properties

Non-functional properties from the end-user perspective are identified and elaborated in the syntactical view. Firstly they are shown in the extended behavioural view and later on validated by the performance simulation. We focus in the sequel on one non-functional property, FrameRate, showing the way it is carried through the design process. This is outlined in Figure 1.

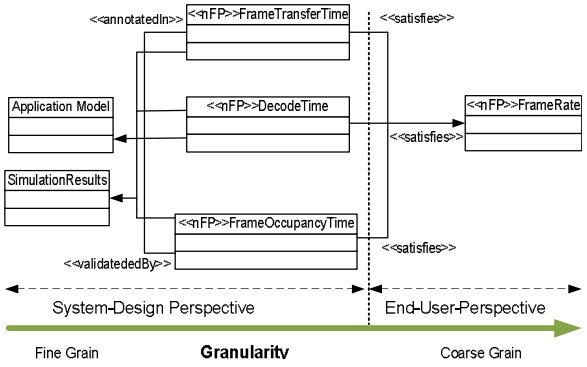

Figure 1: Carrying FrameRate through the application modelling and performance simulation process

3.3. Case Study

Mobile Video Player (MVP) Application: The modelling phase was initiated by splitting the application model into a set of views defined by GENESYS. Then they were modelled using UML2.0 MARTE [19] profile and Papyrus UML tool [19].

3.3.1. Use case view. The use case view shows a system level capability PlayVideo(), which is refined to platform level services requested by the application from the underlying execution platform, e.g. *DecodeVideoClipSegment* as shown in Figure 2.

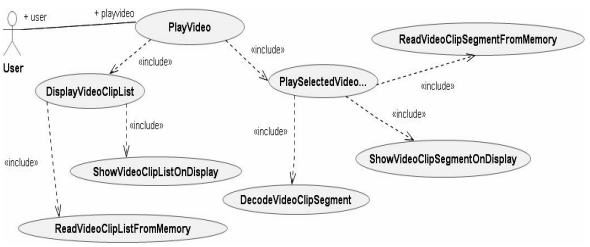

Figure 2: Use case view showing system and platform level services

3.3.2. Structural View. MVP application has four sub-systems, whose services define interfaces for application development. They are stereotyped as <<subsystem>> and <<component>> using the MARTE profile as shown in Figure 3 and Figure 4.

Services are linked to the corresponding sub-systems using MARTE <<allocated>> stereotype.

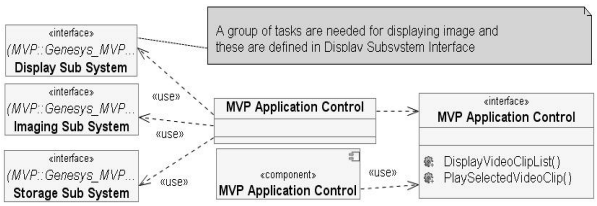

Figure 3:Sub-systems providing services to the application.

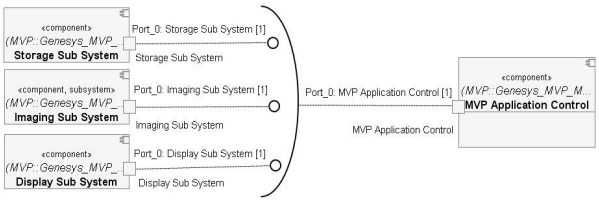

Figure 4: Component diagram showing interfaces of sub-systems.

3.3.3. Syntactical View. The four servers (sub-systems) admit different messages and are stereotype as <<RtUnit>> of the UML2.0 MARTE profile, indicating a computational resource in the execution platform as shown in Figure 5. Non-functional properties are assigned values in slots and could be shown as tagged values as in Figure 6.

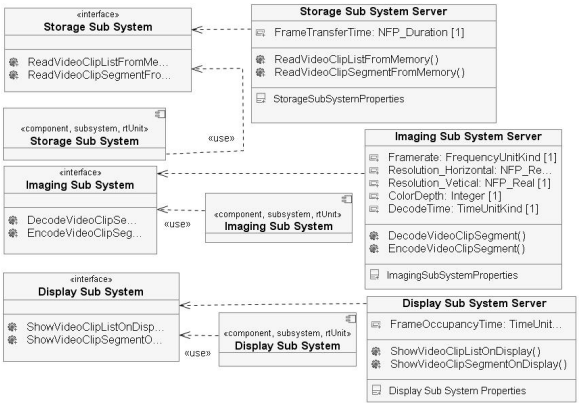

Figure 5: Syntactical View of MVP application.

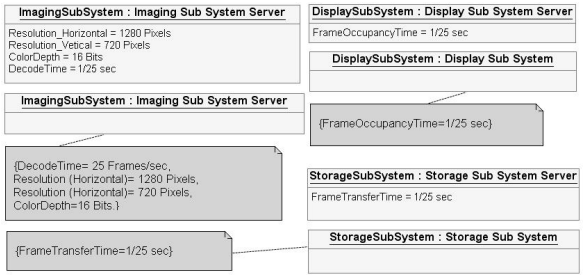

Figure 6: Non-functional properties annotated in slots.

3.4.4. Behavioural View. The behavioural view uses stereotypes of the high-level application modelling (HLAM) sub-profile of MARTE, e.g. <<rtFeature>> and <<RtAction>> in UML state and activity diagrams to describe the behaviour of services and applications as depicted in Figure 7.

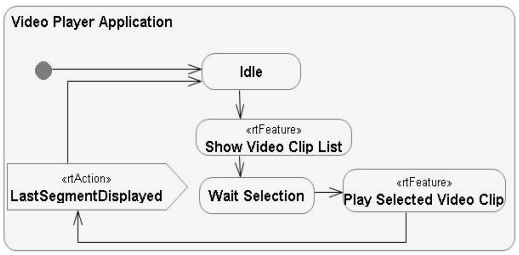

Figure 7: Main operation of MVP application

4. Performance Analysis Approach

The performance evaluation approach follows the Y-chart model [5] consisting of application workload and platform model [8]. After mapping the workloads to the platform, the models are combined for transaction-level performance simulation in SystemC. Based on the simulation results, we can analyze e.g. processor utilization, memory traffic and execution time.

The approach enables performance evaluation early, exhibits light modelling effort, allows fast exploration iteration, reuses application and platform models, and provides performance results that are accurate enough for system-level exploration.

4.1. Application Workload Model

The application workload model has a layered architecture as explained in [8]. The hierarchical structure of the application workload model is shown in Figure 8.

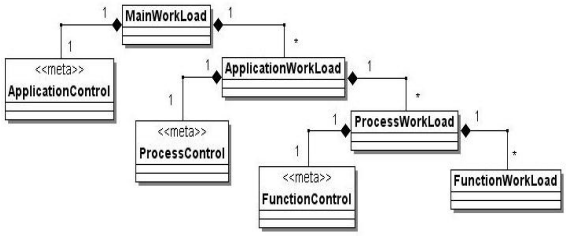

Figure 8: Hierarchical structure of application workload model.

4.2. Platform Model

The platform model is an abstract hierarchical representation of actual platform architecture. It is composed of three layers: component layer, subsystem layer, and platform architecture layer as shown in Figure 9. Each layer has its own services, which are abstract views of the architecture models. Services in subsystem and platform architecture layers are invoked by application workload models.

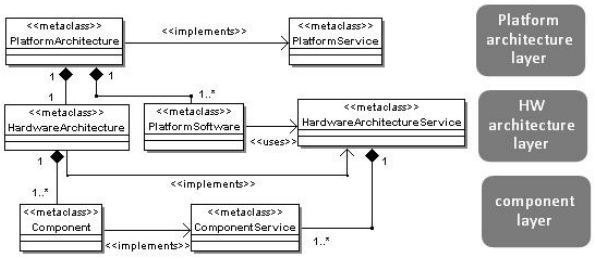

Figure 9: Platform model layers.

The platform model provides two interfaces. The low-level interface transfers load primitives and the high-level interface enables workload models to request services from the platform model. Operating system (OS) models control access to the processing unit models of the platform by scheduling the execution of process workload models.

5. Performance Modelling of GENESYS Applications

For performance modelling of GENESYS applications, workload models are created with UML by mapping from the extended behavioural view of the GENESYS model to the layered workload model, and followed by transformation to SystemC as shown in Figure 10. The platform modelling and mapping between the application and platform models are the same as described in Chapter 4.

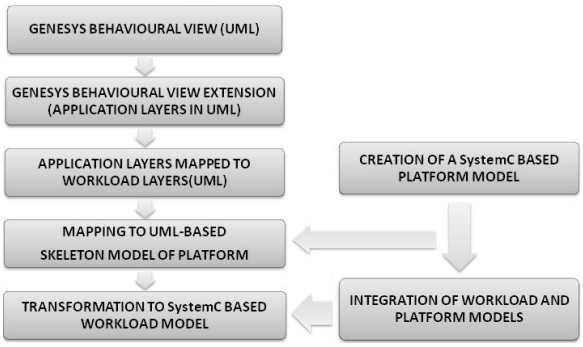

Figure 10: The performance simulation approach.

5.1. Extended GENESYS behavioural view

GENESYS behavioural view is extended to extract a layered hierarchical structure of applications. The first layer defines one or more applications supported by the embedded system to satisfy different use cases.

$$Ea = \{C_E, A1, A2, \ldots, An\}, (1)$$

where *A1*, *A2*,…, *An* are different applications and C_E is the control.

In the second layer each application is refined in terms of used platform services .i.e.,

$$Ai = \{C_A, S1, S2, \ldots, Sn\}, (2)$$

where C_A is the control between services *S1*, *S2*, …, *Sn*.

The MVP application control C_A is shown as an activity diagram in Figure 11. Services requested by the application from the execution platform are stereotyped with <<allocated>> from MARTE profile.

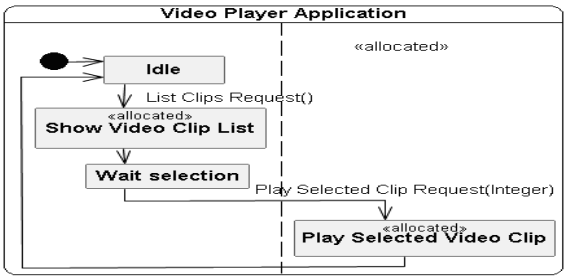

Figure 11: The video player application control.

In the third layer each platform service is a composition of one or more processes. Each process runs on one specific sub-system .i.e.,

$$Si = \{C_S, P1, P2, \ldots, Pn\}, (3)$$

where *Cs* is the control between processes *P1*, *P2*,. . ., *Pn*.

Figure 12 shows *PlaySelectedVideoClip()* service of the MVP application and its associated processes. The activity diagram itself describes the control *Cp*. The <<rtFeature>> stereotype of MARTE adds timing information to actions. Each process is stereotyped with <<allocated>> MARTE stereotype since a single process is confined to one and only one sub-system.

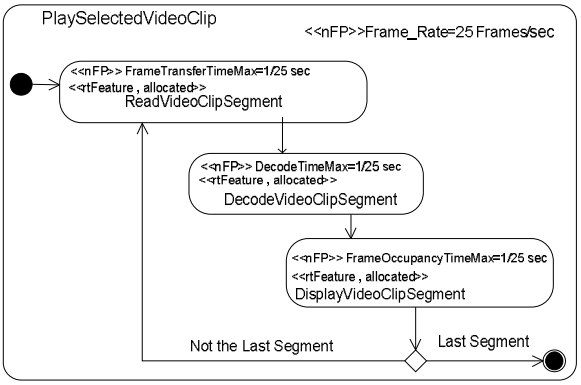

Figure 12: PlaySelectedVideoClip service execution control -with refined non functional properties.

Furthermore, the non functional property, FrameRate is further refined to three non-functional properties from the design perspective. Due to using pipelining in the processing, each of the frame transfer, decoding and display operations has to be

performed within 1/25 seconds (the required frame rate).

In the fourth layer each process is composed of intra sub-system service calls .i.e. $Pi = \{C_P, C1, C2, \ldots, Cn\}, (4)$,

where C_P is control. One such process is shown in Figure 13.

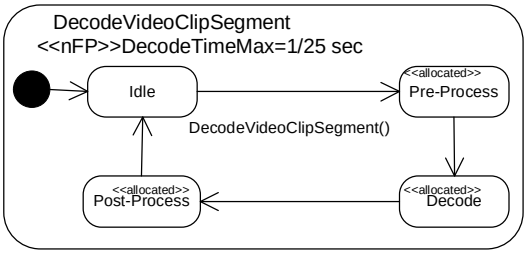

Figure 13: Video clip decoding process (Imaging-sub-system).

A sub-system service call invokes functions (intra sub-system) to properly handle the service call. If C_F is the control, we can express layer 5 as

$$Ci = \{C_F, F1, F2, \ldots, Fn\}, (5)$$

5.2. Extracting Workload Layers from Extended GENESYS Application Layers

The mapping between the GENESYS application layers and the workload model layers is shown in Figure 14.

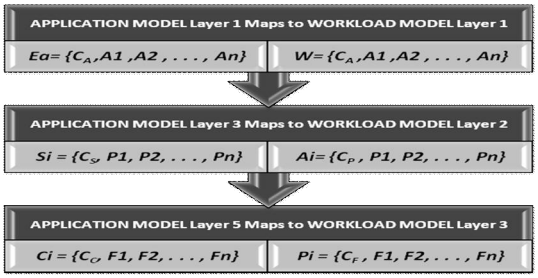

Figure 14: Mapping GENESYS Application to Workload Layers.

The word *mapping* does not apply to the transformation of workload models automatically from the application models but provides a structure for the workloads by providing a certain workload model element corresponding to a particular application model element.

6. Performance Simulation and Results

The platform model is the same as in [8] consisting of four sub-systems interconnected by a ring network with best-effort (BE) and guaranteed throughput (GT) routing approaches. Internally sub-systems can be considered as bus-based specialised, small computers described briefly as follows.

The General purpose (GP) sub-system contains two ARM11 processors, SDRAM and a network interface. It executes the MVP application and hosts operating system.

The Imaging (IM) sub-system contains an ARM7 processor, SRAM, video accelerator, DMA controller and network interface. It provides services for hardware-accelerated image processing, video playback and recording.

The Storage (ST) subsystem contains ARM7, SRAM, SDRAM, DMA controller and a network interface. It maintains a repository for video clips and provides services for loading and storing video clips.

The Display (DP) sub-system contains ARM7, SRAM, Display interface, DMA controller and a network interface. It displays frame buffer data on a screen.

The workload models are from mapping between GENESYS application model layers and workload layers and they were mapped to the aforementioned platform model. The system works in the performance simulation as follows: The MVP application executed in one of the ARM11 processors of the GP sub-system requests a list of movie files from the ObjectServer in ST subsystem. Once a movie file is selected, the MVP Application triggers the VideoPlayer in IM sub-system. MVP application requests movie file from ObjectServer, initiating streaming of the file. The VideoPlayer decodes the compressed stream and transfers video frames to DisplayServer in DP sub-system. DisplayServer displays video frames on screen.

The platform model is equipped with status probes, timers and counters to collect performance data during the simulation. Table 1 shows the utilization of different platform components, while Table 2 shows average values of processing times of some platform services.

Component	GP	ST	IM	DP
ARM0	11%	20%	10%	15%
ARM1	51%			
Video_accel			60%	
DMA_controller		5%	0%	6%
Display_IF				31%
Network_IF	1%	3%	2%	5%
Bus	14%	10%	7%	24%
SDRAM	16%	3%		
SRAM		2%	3%	18%

Table 1. Utilization of platform components.

Table 2. Examples of processing times of services.

Non-Functional Property	Service	Sub-sys	Avg
	dma_transfer	ST	2.4ms
FrameTransfer-TimeMax=1/25 sec	memcpy	ST	12 ms
DecodeTimeMax =1/25 sec	video_decode	IM	14 ms
	dma_transfer	DP	3.1 ms
FrameOccupancy-TimeMax=1/25 sec	memcpy	DP	10 ms

The non functional property FrameRate can be validated from the above simulation results by observing that the system works on video frames in a pipelined fashion and FrameRate was refined in Figure 12 to three sub-properties. The services and corresponding non-functional properties are shown in Table 2. Observing the related services and corresponding non-functional properties side by side, we note that all non functional properties are satisfied. In other words they all are on the average performed within 1/25 seconds.

7. Conclusions

The GENESYS application modelling methodology was extended by linking it with the workload modelling used in performance evaluation. Based on the different views involved in the GENESYS, the behavioural view was extended to obtain a layered application model that can be mapped to the workload model layers used in the applied performance simulation approach.

The approach was experimented in a mobile video player case study, where the application models were created using the GENESYS methodology approach, UML2.0 MARTE profile as the modelling language and Papyrus toolset. The experiments show that considerable reduction of modelling efforts and time could be achieved.

8. Acknowledgements

This work was performed in the Finnish DIEM project partially funded by Tekes, and in the Artemis SOFIA project partially funded by Tekes and the European Union. The authors gratefully acknowledge the European Union projects GENESYS Nr 213322 and MOSART Nr 215244 that provided the information on the application modelling methodology and performance simulation approach, respectively. We would like to thank Adrian Noguero from European Software Institute (ESI) for expert advice on using the MARTE profile and the Papyrus tool.

9. References

[1] T. Noergaard. *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers.* ELSEVIER, UK; 2005, 640 p.

[2] K. Keutzer, A. Newton, J. Rabaey and A. Sangiovanni-Vincentelli,"System-level design: orthogonalization of concerns and platformbased design", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 19 (12), 2000, pp. 1523 - 1543.

[3] A. Sangiovanni-Vincentelli. Defining Platform-Based Design. www.eedesign.com/story/OEG20020204S0062, EEDesign. February 2002.

[4] J. Arlow, Ila Neustadt. *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design.* 2nd Edition. Addison-Wesley, USA; 2008, 592 p.

[5] B. Kienhuis, E. Deprettere, K. Vissers and P. van der Wolf. Approach for quantitative analysis of application-specific dataflow architectures. The IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP '97), pp. 338–349, Zurich, Switzerland. July 1997.

[6] P. Boulet, J. Dekeyser, C. Dumoulin, and P. Marquet. Mda for soc design, intensive signal processing experiment. The Forum on Specification and Design Languages, ECSI, 2003.

[7] http://www.genesys-platform.eu

[8] J. Kreku, M. Hoppari, T. Kestilä, Y. Qu, J.-P. Soininen, P. Andersson and K. Tiensyrjä. Combining UML2 Application and SystemC Platform Modelling for Performance Evaluation of Real-Time Embedded Systems. Hindawi Publishing Corporation. EURASIP Journal on Embedded Systems. Volume 2008, Article ID 712329, 18 pages, doi:10.1155/2008/712329.

[9] L. Bass, P. Clements and R. Kazman. Software Architecture in Practice. Addison-Wesley, 1999, 528 p.

[10] http://www.uml.org

[11] USC Center for Software Engineering, Guidelines for Model-Based (System) Architecting and Software Engineering, http://sunset.usc.edu/research/MBASE, 2003.

[12] D. Garlan, R. T. Monroe and D. Wile. Acme: An Architecture Description Interchange Language. Proceedings of CASCON '97, November 1997.

[13] R. Roshande, B. Schmerl, N. Medvidovic and D. Garlan, D. Zhang. Understanding tradeoffs among different architectural modeling approaches. Software Architecture, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference on software architecture.12-15 June 2004, pp. 47 – 56

[14] P. Lieverse, P. van der Wolf, K. Vissers, and E. Deprettere, A methodology for architecture exploration of heterogeneous signal processing systems. Kluwer Journal of VLSI Signal Processing 29 (3), 2001, pp. 197-207.

[15] A. Pimentel and C. Erbas. A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels. IEEE Transactions on Computers, vol. 55, no. 2, Feb. 2006, pp.99 –112.

[16] T. Wild, A. Herkersdorf and G.-Y. Lee. TAPES—Trace-based architecture performance evaluation with SystemC. Design Automation for Embedded Systems, Vol. 10, Numbers 2-3, Special Issue on SystemC-based System Modeling, Verification and Synthesis, 2006, pp 157-179.

[17] J.M. Paul, D.E. Thomas, and A.S. Cassidy. High-Level Modeling and Simulation of Single-Chip Programmable Heterogeneous Multiprocessors. ACM Transactions on Design Automation of Electronic Systems, Vol. 10, No. 3, 2005, pp. 431-461.

[18] E. Ovaska, A. Balogh, S. Campos, A. Noguero, A. Pataricza, K. Tiensyrjä and J. Vicedo. *Model and Quality Driven Embedded Systems Engineering* VTT, Espoo, 2009, 208 p.

[19] www.omgmarte.org

[20] http://www.papyrusuml.org

PUBLICATION II

PUBLICATION III

PUBLICATION IV

Analyzing Transport and MAC Layer in System-Level Performance Simulation

Subayal Khan, Jukka Saastamoinen,
Mikko Majanen, Jyrki Huusko
VTT Technical research Center of Finland,
FI-90570, Oulu, Finland
email:{subayal.khan,jukka.saastamoinen,
mikko.majanen,jyrki.huusko}@vtt.fi

Jari Nurmi
Tampere University of Technology,
Department of Computer Systems
P.O.Box 553 (Korkeakoulunkatu 1),
FIN-33101 Tampere, FINLAND
jari.nurmi@tut.fi

Abstract:
The modern mobile embedded devices support complex distributed applications via heterogeneous multi-core platforms. For the successful deployment of these applications, the scalability and performance analysis must be performed at all the layers of OSI model. This helps to identify the potential bottlenecks at different layers to perform the necessary optimizations. To achieve this goal, a framework is needed which accurately models the functionalities at different layers. The technical contributions described in this article include the extensions of ABstract inStruction wOrkLoad & execUtion plaTform based performance simulation (ABSOLUT) for the performance and scalability analysis of Transport and Medium Access Control (MAC) layers in the system level performance simulation. The article elaborates the design accuracy of the modeled components and their application in the context of M3 (multi-device, multi-vendor, multi-domain), which is a tri-layered conceptual interoperability architecture for embedded devices. These extensions pave the way towards the full coverage of the OSI model in the system-level performance simulation of distributed embedded systems. The network simulators for example ns-2, OMNeT++ and OPNET though provide detailed models of transport and MAC protocols but do not provide any framework such that these models can be used by the application workload models to mimic the real world use-cases. Also these models do not model the execution workload of these protocols on a particular execution platform and hence cannot be used in the architectural exploration of distributed embedded systems.

I. INTRODUCTION

Modern nomadic devices support computationally intense distributed applications by using heterogeneous multiprocessor architecture platforms. Deployment of a new distributed application is challenging not only due to the heterogeneous parallelism in the platforms [1], but also due to performance and energy constraints. Furthermore, these devices employ diverse communication, transport technologies and application-level protocols to enable information sharing and synchronization among the processes of distributed applications. These technologies enable complex use-cases spanning multiple devices. To evaluate the feasibility of these use-cases, the system-level performance simulation methodology must identify the potential bottlenecks at each layer of the OSI model so that the appropriate optimizations can be performed. The performance analysis of protocols spanning different layers of OSI model require a framework which models these protocols with reasonable accuracy while maintaining a good simulation speed.

In case of non-distributed applications, the processes use inter-process communication (IPC) for synchronization of tasks and the transport, Datalink and Physical layers of the OSI-Model do not play any role. The system-level performance simulation of non-distributed application requires application workload models, platform capacity models and workload models for external libraries. ABSOLUT performance simulation methodology has been already used to evaluate such use-cases [2] and [3].

In the case of distributed applications, the use-cases span multiple devices and the processes communicate with each other via a transport API. The transport layer APIs are implemented on top of Layer 2 of the OSI model which in turn makes use of physical layer. Therefore apart from the performance evaluation of the platform components, the performance evaluation of the applied transport technology, MAC protocol and transmission techniques must be performed. Afterwards, the performance of same application level workload models can be evaluated with other available alternatives of transport, MAC and transmission technologies. The design space is thus much bigger and spans all layers of the OSI model.

The main contribution of this article is to describe the extensions made to ABSOLUT for the performance evaluation of distributed applications. A detailed survey of performance simulation techniques has been presented in [4], therefore the landmark performance simulation methodologies are not discussed in this article.

Rest of paper is organized as follows: Section 2 briefly explains ABSOLUT methodology. Section 4 describes the extensions made to ABSOLUT for enabling the analysis of Layer-2 in System-level performance evaluation. We first elaborate a general method for developing operating system (OS) services for Transport, MAC and physical layers of OSI models. These extensions pave the way towards the full coverage of the OSI model in the system-level performance simulation. These services are implemented by using freely available tools and libraries such as SystemC [5] and itpp library [6]. Afterwards we focus on the modeling and integration of IEEE 802.11 MAC and transport layer of OSI model to ABSOLUT. In section 5, the modeled components are used for the MAC and transport layer scalability and performance analysis of M3 [7]. The obtained simulation results are compared with the ns-2 [8] and OMNeT++ [9] network simulators under different simulation scenarios. Conclusions and Future work are mentioned in Section 6.

II. ABSOLUT

ABSOLUT follows the Y-chart model consisting of application workload and platform model [10]. The workload models are mapped to the platform for transaction-level performance simulation in SystemC [2].

3.1 Application Workload Model

The workloads consist of four layers .i.e., main workload, application workload, process workload and function workload as shown in Figure 1.

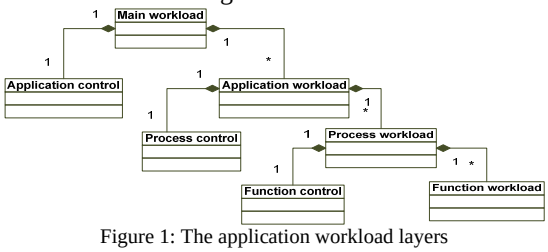

Figure 1: The application workload layers

3.2 Execution Platform Model

The platform model is an abstract hierarchical representation of actual platform architecture. It is composed of three layers: component layer, subsystem layer, and platform architecture layer as shown in Figure 2. Each layer has its own services, which are abstract views of the architecture models. Services in subsystem and platform architecture layers are invoked by application workload models.

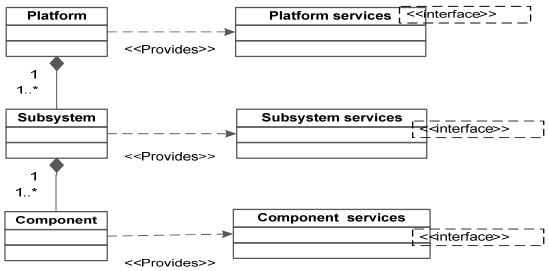

Figure 2: The platform architecture model layers.

III. EXTENDING ABSOLUT FOR ANALYSIS OF LAYER-2

The performance modelling of distributed applications via ABSOLUT demands the modelling and integration of Transport and MAC protocols, modulation techniques, coding schemes and channel models. We now elaborate the modelling and integration of these components to ABSOLUT.

3.1 Design and integration of OS Services

Extension of ABSOLUT for analysing the performance of protocols operating at different layers of OSI model during system-level performance simulation for architectural exploration demands a mechanism for instantiating new H.W and S.W services. These services are registered to the operating system and are used by the application workload models. Furthermore the services operating at a higher layer for example transport-level services (such as TCP) use Data-link level services such as IEEE 802.11 MAC protocols for the transmissions of frames of a packet. These services are created by deriving them from the *OS_Service* base class as shown in Figure 3 which implements the *Generic_Serv_IF*.

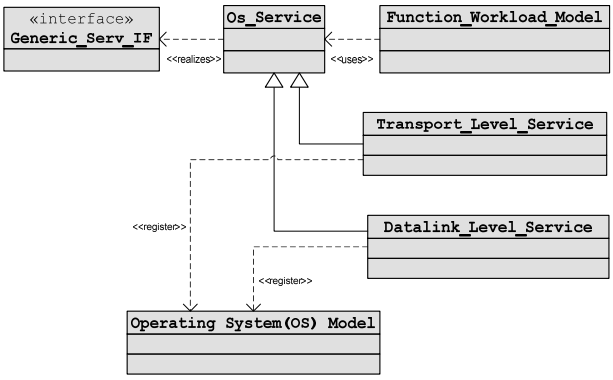

Figure 3: OS services implemented to model use cases spanning multiple devices and for modeling BSD API as OS services.

Implementation of *Generic_Serv_IF* by the *OS_Service* base enables the process-level application workload models or higher level services request a certain service by its name and invoke the functionality implemented by the derived service. This is shown in Figure 4.

```
SID=Use_Service(”Serv_Name”);
Wait_Service(SID);
```

Figure 4: Accessing an OS_Service via Generic_Serv_IF

3.2 Accessibility and Hierarchy of *OS_Services*

The *OS_Service* base class implements the functionality related to scheduling of service requests of processes via priority queues and informs the requesting process on service completion after taking it to running state again. This is shown in Figure 5.

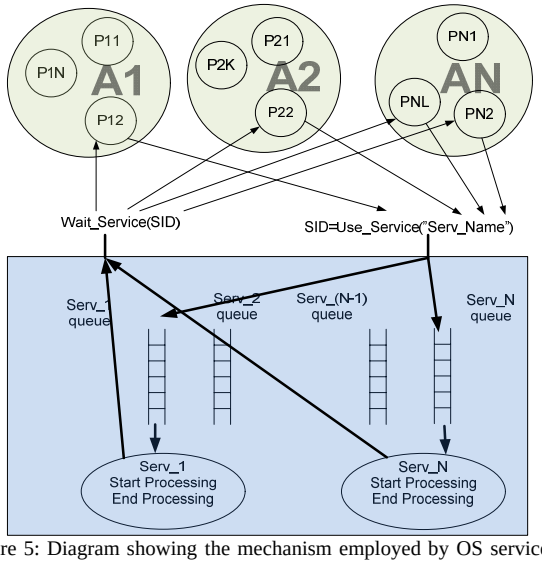

Figure 5: Diagram showing the mechanism employed by OS services to execute requests of processes.

The derived services implement the service-specific functionality making service modelling straight forward. The services at upper layers make use of services at lower layer .e.g., Transport layer use Data-Link layer services.

The services are accessed from the platform using the service name assigned while registration to the OS model. For example BSD socket API function “send()” can be modelled as an *OS_Service* and registered by a unique service name for example “PktTx” to the OS. It can then accessed by the process workload models by using its unique service name via *Generic_Serv_IF*. This is shown

in Figure 4 and Figure 6.

```
//The processor model
ARM_Crtx_Proc_ptr=new Scalable_MultiCore_CPU
"m_ARMCortex_A9_MP_Processor");

//Creating the operating system (OS) model
m_os = new Generic_serv_op_sys
("os",ARM_Crtx_Proc_ptr->GetProcessorCores(),
m_os_addr);

//Send() API function registered as "PacketTx" to OS
Pkt_Tx_Service= new Packet_Tx_Serv("Transmit_Packet",m_os);
Serv_type Msg_serv_type = { SERV_TYPE_LOCAL };
m_os->register_service(Pkt_Tx_Service,"PacketTx",
Msg_serv_type);

//This service handles transmission of single frame via
//IEEE 802.11 DCF. Registered by name "FrameTx" to OS
Frame_Tx_Service= new Frame_Tx_Serv("Transmit_Frame",m_os);
Serv_type Frame_serv_type = { SERV_TYPE_LOCAL };
m_os->register_service(Frame_Tx_Service,"FrameTx",
Frame_serv_type);
```

Figure 6: Registration of services to the operating system (OS) model

3.3 Implementation and integration of MAC and Transport Level OS_Services

IEEE 802.11 distributed coordination function (DCF) requires a station wanting to transmit, to first listen to the channel to check its status (occupied or not) for a DCF Interframe Space (DIFS) interval. If the channel is found busy during the DIFS interval, the station defers its transmission. In a network where a number of stations contend for the wireless medium, if multiple stations sense the channel busy and defer their access, they will also virtually simultaneously find that the channel is released and then try to seize the channel. As a result, collisions may occur. In order to avoid such collisions, DCF also specifies random back off, which forces a station to defer its access to the channel for an extra period. DCF also has an optional virtual carrier sense mechanism that exchanges short Request-to-send (RTS) and Clear-to-send (CTS) frames between source and destination stations during the intervals between the data frame transmissions. The IEEE 802.11 DCF can be shown in the form of a flow chart as in Figure 7.

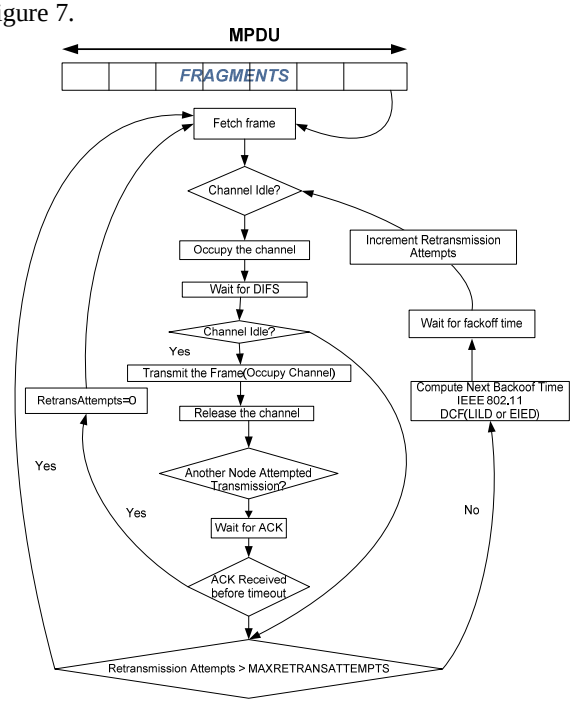

Figure 7: Flow chart of IEEE 802.11 DCF

The IEEE 802.11 DCF is implemented by the following M3_Frame_Tx class derived from the *OS_Service* base class. Only the constructor of the class is shown. The DIFS, SLOT_TIME [11] and the initial value of the contention window are assigned in the constructor as shown in Figure8.

```
M3_Frame_Tx::M3_Frame_Tx
(sc_module_name _name,
Proc_ctl_IF * host)
:OS_Service(_name){

//Initializing values of

//Contention window
CwCurrent=CWMIN;

//DIFS
DIFS=sc_time(128,SC_US);

//Slot Time
SLOT_TIME=sc_time(50,SC_US);

//Backoff time
BackOffTime=SLOT_TIME;

//Retransmission attempts
RetransmissionAttempts=0;
}
```

Figure8: The Frame Transmit Service initializing MAC parameters

In ABSOLUT the MAC protocols as well as transport-level protocols such as TCP are also modeled as servies by deriving them from the same base class *OS_Service* which provides the scheduling and synchronization mechanism. The transport-level services then request the Layer-2 services such as IEEE 802.11 for frame transmission. The do_service() method of the *OS_Service* base class is implemented by the derived class to provide the functionality of IEEE 802.11 DCF as shown in Figure 10. The do_service() method spams a separate frame transmission function for handling each request of frame transmission from the transport as shown in Figure 9.

```
void M3_Frame_Tx::do_service(){
while(true)
 {

 //Get Service attributes from OS
 m_crnt_attr=
 const_cast< Serv_attributes *>
 Current_Service_Attributes);

 //Initial retransmission Attempts
 RetransmissionAttempts=0;

 //Spawn Frame Transmission Function
 SC_FORK
 sc_spawn
 sc_bind(&M3_Frame_Tx::Tx_Crnt_Frame,
 this,static_cast<Smart_M3_Attr *>
 m_crnt_attr)))
 SC_JOIN

 //Inform OS about service completion
 m_service_complete_ev.notify();

 //Wait for next Frame Tx request
 wait();
 }
}
```

Figure 9: Implementation of Frame Transmit service. It invokes a spammed function for the transmission of a single frame.

The spammed function implements the flow chart shown in Figure 7. It attempts the retransmission transmission for up to a maximum number of retransmissions, the frame is considered lost and the transport-level service (transport protocol) is informed. For connection oriented protocols, the remaining frames are not transmitted. For connection-

less protocols this information is neglected.

The transport-Level services .i.e., TCP and UDP are also modeled as *OS_Services*. Both the services make use of MAC level services for frame transmission. TCP calls the Frame Transmission service of MAC (which receives a single frame at a time for transmission) as many times as the number of frames in the transport packet. If one frame is lost (due to errors or collisions) the MAC informs transport and the rest of the frames are dropped and a packet loss is recorded. In the UDP transport-level service, the MAC does not inform the transport layer about the frame loss and all the frames are transmitted even if one or more frames of a packet are lost (due to collisions or errors). The accuracy of the modelled components is elaborated in the next section.

IV. ACCURACY OF SIMULATION RESULTS

To study the MAC protocols in isolation under a particular scenario, we abstract out the Application workloads with delays obtained after profiling or use traffic generators. Three types of traffic generators .i.e., pareto on off, exponential and constant-bit rate available in ns-2 have been integrated to ABSOLUT. The different modulation techniques like QPSK and BPSK have been modeled along-with MC-CDMA. Two channel coding techniques .i.e., convolutional and Reed Solomon codes and two channel models .i.e., binary symmetric channel and additive white Gaussian noise (AWGN) channels have been integrated by using models available in itpp library. The performance model is configured with a certain type of modulation scheme, coding scheme and channel model. Bit errors are computed using the functions available in itpp library. Frame lengths can be chosen randomly or fixed to a value before simulation to analyze MAC and transport protocols in a particular scenario.

4.1 Analysing accuracy of bit error rate calculation

Different modulation schemes available in itpp library have been used without modification. We present the results for Multi-Code CDMA with QPSK modulation. For $1e^6$ bits the results are over 99.8% accurate (when compared to theoretical results) as shown in Figure 10.

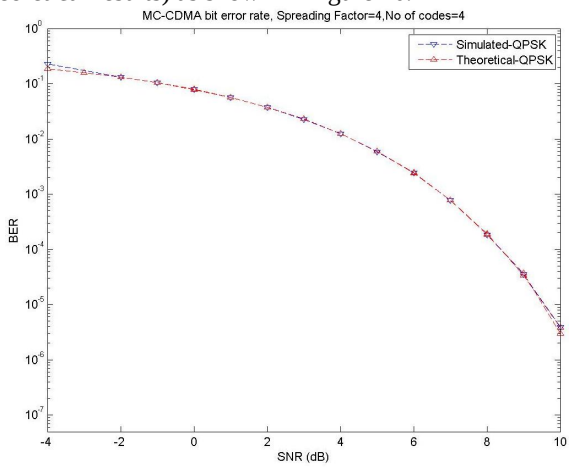

Figure 10: Theoretical versus simulation bit error rate for MC-CDMA with QPSK. Number of codes (M) = 4 .Spreading Factor (k=4). Number of bits =100,000.

4.2 Analysing accuracy of frame error rate calculation

In the absence of any encoding in IEEE 802.11, the fragment and the bit error rate are related by Equation 1.

$$P_e = 1-(1-BER)^S \quad (1)$$

Where *s* is the fragment size and *BER* is the Bit Error Rate and *Pe* is the probability of frame error. The bit error rates are plotted against frame error rates for different values of frame lengths as is shown in Figure 11.

The frame and bit-error rates can be recorded directly from simulation and plotted for different values of bit error rates as shown in Figure 11. The recorded simulation results are over 92% accurate when averaged after 20 simulation runs. The simulation results are compared to analytical results for Packet Lengths of 228 and 2228 as shown in Figure 11.

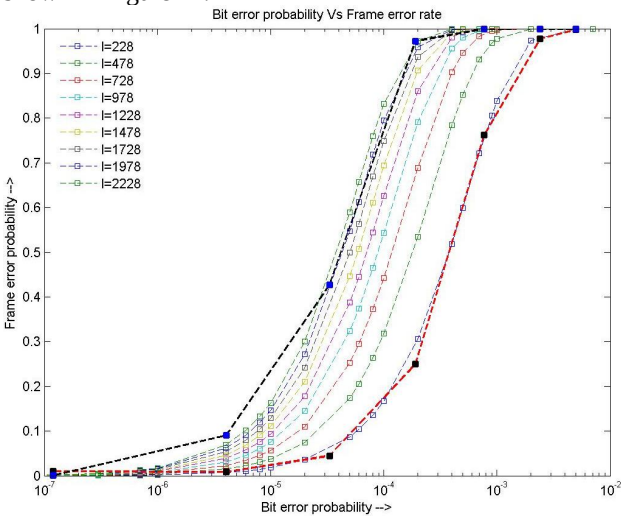

Figure 11: Frame error probability versus bit error rate. Theoretical results compared to simulation results for frame lengths 228 and 2228.

4.3 Analysing accuracy of packet error rate

In case of IEEE 802.11, one MAC service data unit (MSDU) can be partitioned into a sequence of smaller MAC protocol data unit (MPDUs) in order to increase reliability. Fragmentation is performed at each immediate transmitter. The process of recombining MPDUs into a single MSDU is called defragmentation. Defragmentation is also done at each immediate recipient. When a directed MSDU is received from the LLC with a length greater than a Fragmentation-Threshold, the MSDU is divided into MPDUs. Each fragment's length is smaller or equal to a Fragmentation-Threshold [11]. The MPDUs are sent as independent transmissions, each of which is separately acknowledged. The loss probability of transmitting a transport packet fragmented at the MAC layer into N fragments is given by the Equation 2 [12].

$$P_{wl} = 1-\left(\sum_{i=1}^{1=M} P_l^{i-1}(1-P_l)\right)^N = 1-\left(1-P_l^{M-1}\right)^N \quad (2)$$

Where P_l denotes the successful transmission probability of one attempt, *i* denotes the retransmission attempts and *M* is the maximum number of retransmission attempts. Figure 12 shows the transport packet loss rate as a function of the MAC frame loss probability during each transmission retry for a fixed number of fragments *(N=10)* and for different values of maximum retransmission attempts[12]

(M=1→10). The simulation results are compared to the analytical results as shown in Figure 12. The values of *M* and *N* were fixed, the value of signal to noise ratio (SNR) was varied and the simulation was repeated several times. The results for each value of SNR were averaged to obtain each point on the two curves. The simulation was run 20 times and the averaged results achieve an accuracy of over 85% when compared with analytical results as shown in Figure 12.

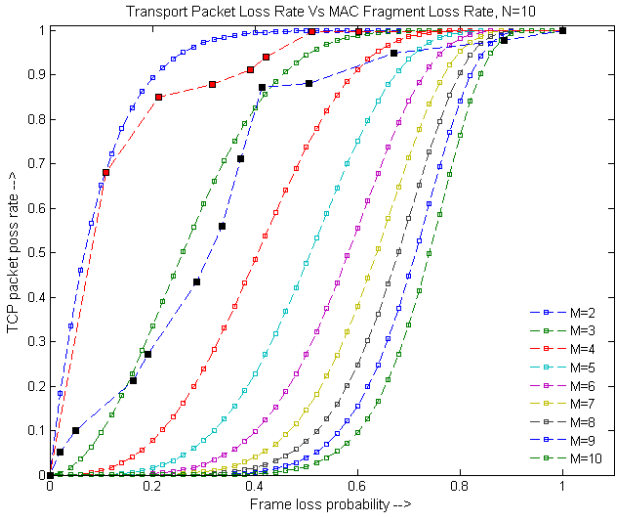

Figure 12: Theoretical versus simulation results. MAC Frame loss probability versus transport packet loss rate, for Maximum Retransmission attempts (M=2 and 3) and number of fragments (N=10).

4.4 Comparing MAC and Transport layer models with ns-2 and OMNeT++ network simulators

We now compare the throughput, Frame delays and Packet delays of ABSOLUT MAC and Transport models with ns-2 and OMNeT++ network simulators. Different Packet lengths, transport protocols and Packet transmission rates are used in both the case studies. The simulations are carried out under saturated conditions. The simulation parameters are mentioned in Table 1.

Table 1: Experiment parameters

Parameters	Values
SIFS	10 micro seconds
DIFS	50 micro seconds
Slot Interval	20 micro seconds
Preamble Length	144 bits
PLCP header Length	48 bits
Channel bit rate	2 Mbps
CWmin	32
CWmax	2048
CWo	32
EW	16

The simulations are carried out in WLAN environment in the context of M3. The abbreviation M3 means multi-device, multi-vendor, multi-domain to highlight the flexibility and portability of the technology [7]. It means that all the network client nodes called Knowledge Processors (KPs) at information level in M3 are within the transmission range of a single server called Semantic Information Broker (SIB) which acts as the only destination for the KPs.

1) Case Study 1: Comparison of simulation results at MAC and Transport Layer with ns-2

In the first case study we compare the results of our MAC and transport models with ns-2. The data traffic is generated using the Constant bit rate (CBR) traffic generator available in ns-2 simulator and the transport protocol is TCP. [12]. The traffic generators can be configured by using the simple interface as shown in Figure 13.

```
//CBR traffic generator parameters

// 2.5 milliseconds
double AverageBurstTime=.0025;
// 1 second
double InterFrameTime  = 1    ;
// 2048 bytes
double SinglePktSize=2048;
// 2 Mega bits per second
double Bitrate=2000000;

//Instantiate CBRtraffic generator
SimConfig::Instance()->SetTrafficGenerator(
CBR_TRAFFIC_GENERATOR,
AverageBurstTime,
InterFrameTime,
Bitrate,SinglePktSize
);
```

Figure 13: An example configuration of the CBR traffic generator. Packet Length=2048 Bytes. Data rate= 2 Mbps. Average Burst Time=.0025 seconds. Inter Frame Space=1 second.

With the same experimental parameters mentioned in Table 1, we perform our simulations in two different frame lengths .i.e., 512 bytes and 1024 bytes. In both the cases, the transport-level packets are not fragmented into multiple MAC frames; therefore we only use the MAC Frame delays and throughput for comparison. The average Delays for both the frame lengths are shown in Figure 14 for different number of active nodes (20→100 KPs and one SIB in smart spaces) in the network (Smart Space).

The ns-2 and ABSOLUT simulations were run 50 times and the average values were computed. The results indicate that if ns-2 is used as a reference bench mark, the results of ABOLUT Mac and transport are 70-80% accurate. The inaccuracy is due to absence of the RTS/CTS mechanism in ABSOLUT models. The results show that ABSOLUT models always produce pessimistic results, .i.e., less throughput and more delays for the same simulation scenario.

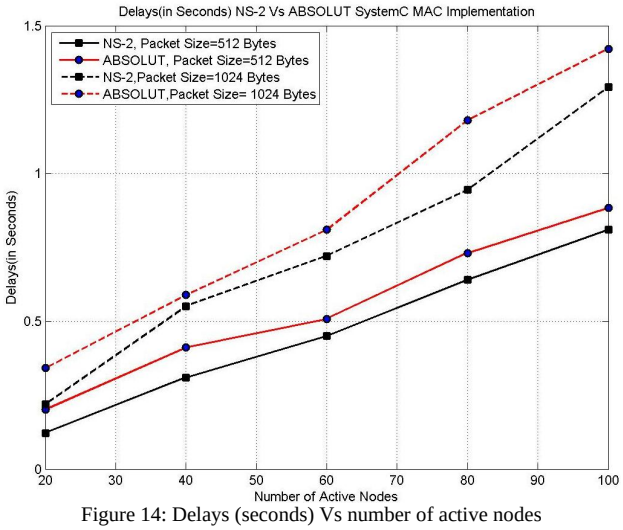

Figure 14: Delays (seconds) Vs number of active nodes (Ns-2 versus ABSOLUT)

The normalized throughput for both the frame lengths is

shown in and Figure 15.

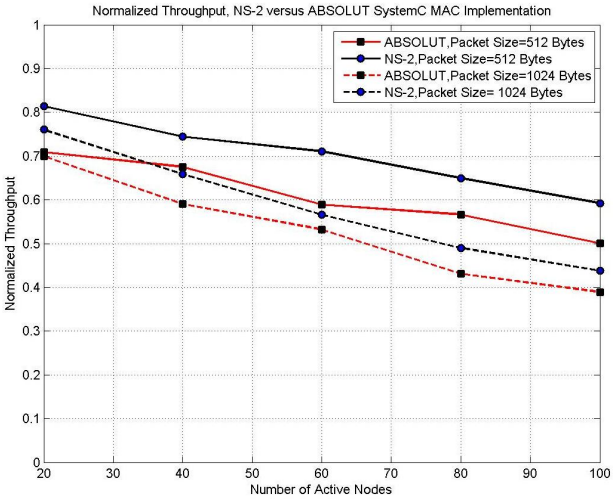

Figure 15: Normalized Throughput versus number of active nodes (Ns-2 Vs ABSOLUT)

The average collisions times (Number of collisions/100 seconds) are shown in Figure 16.

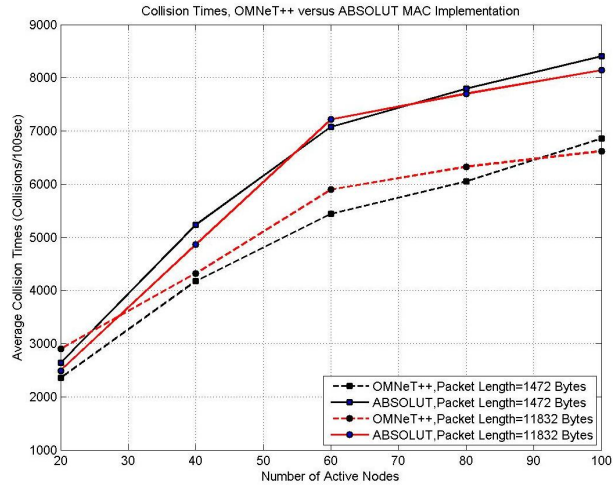

Figure 16: Average collision times Vs number of active nodes (Ns-2 Vs ABSOLUT).

2) Case Study 2: Comparison of simulation results at MAC and Transport Layer with OMNeT++

In the second case study, we compare the results of ABSOLUT MAC and transport models with OMNeT++. No traffic generators were used. The application sends packets at 2 milli-second interval. The simulations are performed under two scenarios. In the first scenario, the application sends 11832 Bytes long packets. The packet is fragmented into 8 fragments. As a consequence MAC sends 8 fragments, each of length 1500 Bytes. In the second scenario, the application sends a 1472 Byte packet at 2 milli-second interval. There is no fragmentation on any layer and as a result, MAC sends a single frame of length 1534 Bytes for each Application packet. Since the packet transmission rate is too fast, the collision rate is quite high which significantly increases the delays and reduces the throughput.

The maximum and average Delays for the packet length of 11832 Bytes(8 Frames/Packet) is shown in Figure 17 as the number of nodes (KPs) is varied (20→100) in the network (Smart Space).The goal is to investigate the case where multiple frames are transmitted for a single transport packet.

The OMNeT++ and ABSOLUT simulations were run 20 times and the average values were computed. The results indicate that if OMNeT++ is used as a reference bench mark, the results of ABOLUT MAC and transport are 75-90% accurate. The inaccuracy is due to the absence of the RTS/CTS mechanism in ABSOLUT models. The results show that ABSOLUT models always produce pessimistic results, .i.e., less throughput and more delays for the same simulation scenario.

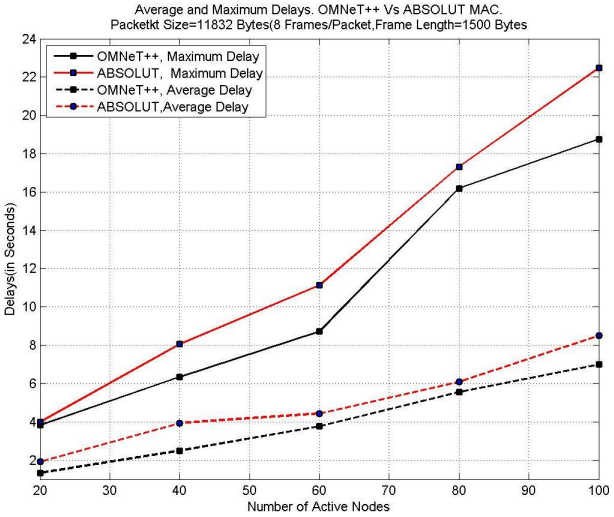

Figure 17: Maximum and Average Delays (seconds) Vs number of active nodes (OMNeT++ Vs ABSOLUT).

The normalized throughput for both the packet lengths is shown in Figure 18.

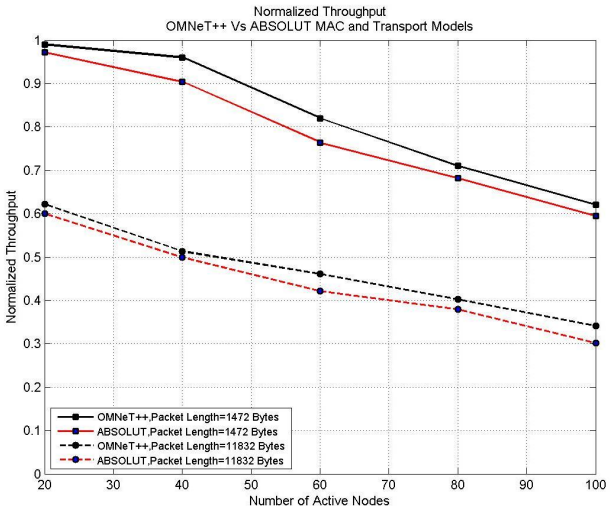

Figure 18: Normalized Throughput Vs number of active nodes (OMNeT++ Vs ABSOLUT).

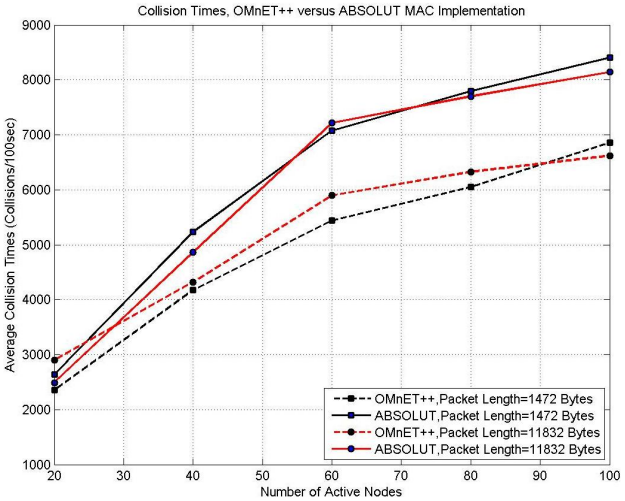

Figure 19: Average collision times Vs number of active nodes (OMNeT++ Vs ABSOLUT).

4.5 Platform utilization of Transport and MAC

Each network node (KPs or SIB) were mapped to a separate ABSOLUT platform model. Each platform model used in the both case studies is a modified OMAP-44x platform model. The MAC and Transport services were registered to the OS model of the platform. The platform model consists of two ARM Cortex-A9 processors consisting of four and three processing cores respectively instead of two (as in case of original TI OMAP44-x platform [13]), SDRAM, a POWERVR SGX40 graphics accelerator and an Image signal processor. This is shown in Figure 20. The Network-on-Chip (NoC) infrastructure was abstracted out and replaced with on-chip bus as shown in Figure 20.

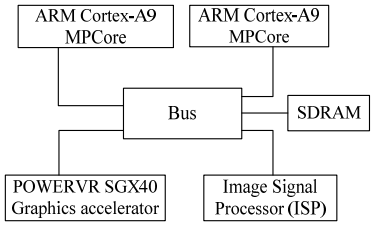

Figure 20: OMAP 44x Platform ABSOLUT model.

Each processor core (Cortex-A9 CPU model) has an L1 and L2 cache and can possibly share an L3 cache with one or more cores in the Multi-Core Processor model. This is shown in Figure 21.

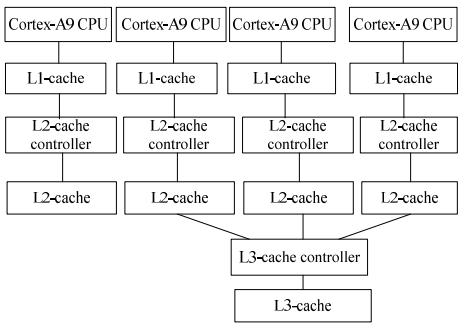

Figure 21: Diagram showing the quad-core processor model used in the performance platform model.

For performance utilization of Transport and MAC layer we only model the workload of the TCP since it is a connection-oriented, complex and more computationally intensive transport protocol then UDP. UDP is a light weight non-connection oriented transport protocol used primarily for real-time streaming applications; therefore processing costs of UDP are lower than TCP.

The workload for TCP *OS_Service* was extracted from [14]. The processing times of TCP functions were scaled down since ARM-Cortex-A9 processors operate at a much faster clock then DECstation 500/200. The approximate number of abstract instructions [2] for the TCP OS_Service processing workload were extracted and modelled as a function workload model [2]. This function workload model mimics the execution workload of the TCP transport protocol. It executes inside a Process workload model triggered by the TCP *OS_Service* during the processing of the service request. The Application models were abstracted out by using constant bit rate traffic generator and constant delays to measure the performance of TCP in isolation.

The average busy time of any processor core of any network node (KP or SIB in the case of M3) involved in the experiment was less than .00001% even for the case of 100 KPs. This is the processing time for TCP. The processing time of IEEE 802.11 is merely a subset of this processing time. The performance costs even after using NoTA DIP over TCP were found to be less than .0001% [15], thus confirming the results.

V. Conclusions and Future Work

The analysis performed in [16] concludes that the network simulators .i.e., ns-2 and OPNET though give different absolute values under the same simulation scenarios but the trend of the results obtained from both the simulators are the same. The ABSOLUT models also validate these results. As the number of KPs is increased, OMNeT++ and ns-2 as well as ABSOLUT show a similar trend in the change in values of delays, throughput and collisions. The following conclusions can be drawn on the basis of the case studies in the previous section.

1. ABSOLUT MAC and Transport models can be used for the system-level performance simulation of distributed embedded systems. In case of distributed applications, the end-end delays and throughput play important role and must be met for providing a pleasant and acceptable end-user experience. The results are reliable since they are pessimistic when compared to state of the art network simulators for example ns-2 and OMNeT++. In other words the values of delays are always higher than OMNeT++ and ns-2. This guarantees that if the ABSOLUT MAC and transport models validate the values of delays and throughput, they are definitely validated by OMNeT++ and ns-2. Thus the modelled MAC and Transport layer services can be confidently used for system-level performance simulation of distributed applications and architectural exploration.
2. The application models can be replaced by traffic generators or constant delays to study the behaviour of Transport and MAC protocols in isolation. This helps to identify the potential bottlenecks at different layers in the OSI model giving full coverage of the OSI protocol stack in

architectural exploration of distributed embedded systems.

3. The performance costs of the MAC and transport layers are negligible and can be abstracted out for system-level performance simulation.

4. In the case of M3 architecture, according to ABSOLUT MAC and transport models, the following conclusions can be drawn in the traffic conditions considered in the two case studies. In first case study, the average delay per packet increases from .24 and .33 seconds to .9 and 1.39 seconds and the normalized throughput decrease from .71 and .7 to .5 and .41 for packet lengths 512 and 1024 bytes. In second case study, the average delay per packet increases from 2 and 4 seconds to 8.1 and 23 seconds and the normalized throughput decrease from .98 and .6 to .6 and .4 for packet lengths 111832 and 1472 bytes respectively. The platform utilization is negligible in both cases. Therefore IEEE 802.11 MAC and TCP do not show any significant performance bottlenecks as far as platform utilization is concerned but the delays increase significantly as more and more KPs join the smart space under the considered traffic conditions. The throughput also decrease significantly as the number of KPs increases in a smart space under these traffic conditions. Hence we conclude from our analysis that for small scale smart spaces such as smart cars and class rooms M3 will operate pretty well with IEEE 802.11 standard operating at Layer-2 and TCP. On the other hand in large scale smart spaces such as smart hockey stadiums operating 1000s of KPs to share information, IEEE 802.11 has to be either optimized or replaced by another potential solution at MAC-layer.

In the future, it is planned to further extend the ABSOLUT methodology to the incorporate multithreading application workload modelling support and C++ workload extraction methodology. These extensions will enable the seamless integration of design and performance simulation of distributed applications. The extended ABSOLUT framework will then employed for the system-level performance simulation of distributed GENESYS, NoTA and M3 applications. In case of non-distributed GENESYS applications that milestone has already been achieved [3].

VI. ACKNOWLEDGEMENTS

This work was performed in the Artemis SOFIA project partially funded by Tekes – The Finnish Funding Agency for Technology and Innovation and the European Union. The work was performed in cooperation with Finnish ICT SHOK research project Future Internet. Authors would like to thank all their colleagues for valuable discussions about the topic.

REFERENCES

[1] K. Keutzer, A. Newton, J. Rabaey and A. Sangiovanni-Vincentelli,“System-level design: orthogonalization of concerns and platformbased design”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 19 (12), 2000, pp. 1523 - 1543.

[2] J. Kreku, M. Hoppari, T. Kestilä, Y. Qu, J.-P. Soininen, P. Andersson and K. Tiensyrjä. Combining UML2 Application and SystemC Platform Modelling for Performance Evaluation of Real-Time Embedded Systems. Hindawi Publishing Corporation. EURASIP Journal on Embedded Systems. Volume 2008, Article ID 712329, 18 pages, doi:10.1155/2008/712329.

[3] Linking GENESYS Application Architecture Modelling with Platform Performance Simulation. Subayal Khan, Susanna Pantsar-Syväniemi, Jari Kreku, Kari Tiensyrjä and Juha-Pekka Soininen. 12th Forum on specification and Design Languages, FDL2009. Sep 22- 24, 2009. Sophia Antipolis, France.

[4] S. Khan et al., “From y-chart to seamless integration of applicationdesign and performance simulation,” in System on Chip (SoC), 2010International Symposium on, 2010, pp. 18 –25.

[5] http://www.systemc.org/home/

[6] http://itpp.sourceforge.net

[7] Lappeteläinen, Antti et. al., 'Networked Systems, Services, and Information - The Ultimate Digital Convergence'. 1st International Conference on Network on Terminal Architecture, June 11, 2008, Helsinki.

[8] http://www.isi.edu/nsnam/ns/

[9] http://www.omnetpp.org/

[10] B. Kienhuis, E. Deprettere, K. Vissers and P. van der Wolf. Approach for quantitative analysis of application-specific dataflow architectures,” in Proceedings of the IEEE International Conference on Application- Specific Systems, Architectures and Processors (ASAP ’97), pp. 338– 349, Zurich, Switzerland. July 1997. USC Center for Software Engineering, Guidelines for Model-Based (System) Architecting and Software Engineering, http://sunset.usc.edu/research/MBASE, 2003.

[11] J.M. Paul, D.E. Thomas, and A.S. Cassidy. High-Level Modeling and Simulation of Single-Chip Programmable Heterogeneous Multiprocessors. ACM Transactions on Design Automation of Electronic Systems, Vol. 10, No. 3, 2005, pp. 431-461.

[12] Fethi Filali, Impact of Link-Layer Fragmentation and Retransmissions on TCP performance in 802.11-based Networks. in Proc. of MWCN 2005, 7th IFIP/IEEE International Conference on Mobile and Wireless Communications Networks, September 19th-21st 2005,Marrakech, Marrocco.

[13] www.ti.com

[14] Jonathan Kay, Joseph Pasquale: The Importance of Non-Data Touching Processing Overheads in TCP/IP. SIGCOMM 1993: 259-268

[15] Instantiation and feasibility evaluation of NoTA SOAD via MARTE profile and binary instrumentation. Khan, Subayal; Tiensyrjä, Kari; Nurmi,Jari. The 7th International Conference and Expo on Emerging Technologies for a Smarter World. CEWIT 2010. Incheon, 27 - 29 Sep. 2010

[16] ns-2 vs. OPNET: a comparative study of the IEEE 802.11e technology on MANET environments, P. Pablo Garrido, Manuel P.Malumbres and Carlos T.Calafate. SIMUTOOLS 2008 - 1st International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems.

PUBLICATION V

PUBLICATION VI

System Level Performance Simulation of distributed GENESYS Applications on multi-core platforms

Subayal Khan, Jukka Saastamoinen, Kari Tiensyrjä
VTT Technical research Center of Finland, FI-90570, Oulu, Finland
e-mail:{subayal.khan,jukka.saastamoinen, kari.tiensyrja }@vtt.fi

Jari Nurmi
Tampere University of Technology, Department of Computer Systems
P.O. Box 553, Korkeakoulunkatu 1, FIN-33101 Tampere, FINLAND
jari.nurmi@tut.fi

***Abstract*-Modern high end mobile devices employ multi-core platforms and support diverse distributed applications due to increased computational power. A brisk performance evaluation phase is required after the application modeling to evaluate feasibility of new distributed applications on the multi-core mobile platforms. GENESYS modeling methodology which employs service-oriented and component based distributed application design has been extended for this purpose such that application level services are refined to platform-level services allowing mapping of GENESYS application architecture to workload models used in performance evaluation. This results in easy extraction of application workload models, reducing the time and effort in the performance evaluation phase needed for architectural exploration. This article presents the way brisk performance evaluation of distributed GENESYS applications is achieved by employing extended GENESYS distributed application architecture. The approach is experimented with a case study. UML2.0 MARTE profile, Papyrus UML2.0 modelling tool and SystemC were used for modelling and simulation.**

Keywords: UML2.0, MARTE profile, GENESYS, ABSOLUT.

I. INTRODUCTION

The rapidly evolving multi-core platforms based mobile devices support diverse and computationally intense distributed applications [1]. The challenges concerning the deployment of such applications are twofold, .i.e., the parallelism in multi-core platforms and the performance and energy constraints.

For efficient product development, it is of pivotal importance that the application design phase of software life cycle provides the foundation for system-level performance evaluation, reducing time and effort involved in architectural exploration. To address such issues as design complexity, time-to-market and first time success, platform-based design was introduced a decade ago [2]. A platform is defined as an abstraction layer facilitating a set of possible refinements into a subsequent abstraction layer in the design flow [3].

In model based software development, primary artefacts of development are models [4]. A model is defined as "a reduced representation of system highlighting properties of interest from a given viewpoint". Models facilitate easier understanding of complex systems and are useful for all the phases of system life cycle.

Y-chart [5] scheme is commonly applied for designing heterogeneous systems, segregating the application and architecture modelling. The application model is mapped onto platform model for analysing properties of the system model [5]. GENESYS application architecture was employed for modelling distributed applications [6]. ABSOLUT [7] is a system level performance evaluation methodology for embedded systems which employs model based approach for both application and platform. The application model is mapped to the platform model for system-level performance simulation. ABSOLUT has been extensively used for the performance simulation of non-distributed embedded applications [7] [8]. For employing ABSOLUT for the performance evaluation of distributed applications, MAC and transport protocol models have been integrated to ABSOLUT framework [9].

So far, the performance evaluation of distributed applications via the extended ABSOLUT framework has not presented. A distributed application runs on two or more devices (computing platforms) which communicate via a computer network. The main contribution of this paper is to present the design and performance evaluation of distributed embedded applications via ABSOLUT in the context of distributed GENESYS applications. The extended ABSOLUT framework can also be used for the performance evaluation of applications designed with other distributed application architectures design methodologies for example NoTA [10].

The rest of the paper is organized as follows: Section 2 gives a brief outline of landmark performance simulation techniques. Section 3 gives an overview of GENESYS SOA and elaborates the modelling of distributed GENESYS applications via a case study. Section 3 elaborates the ABSOLUT performance simulation approach via a case study. Section 4 briefly describes ABSOLUT performance simulation approach. Chapter 5 elaborates the extensions made to ABSOLUT for performance evaluation of distributed applications. It also shows extensions made to GENESYS for integration with performance evaluation phase. Section 6 and section 7 show simulation results and conclusions respectively.

II. RELATED WORK

Software architecture expresses the system in the form of different views, each representing a different aspect of the system [12]. Object Management Group (OMG) defines Model Driven Application Architecture (MDA) relying on efficient use of system models to facilitate transformations between different model types [4]. Unified Modelling Language (UML) is used to describe application structure, behaviour, and architecture [13]. Various Architectural

Description Languages (ADLs) have been proposed. MBASE provides integrated models for capturing the product success, process and properties [14]. Acme design language relies on a core ontology comprising of seven elements representing architectural elements [15]. Mae [16] triggers the modelling, analysis, and management of different versions of architectural artefacts supporting domain-specific extensions to capture other system properties.

Performance modelling has been approached in different ways. SPADE [17] treats applications and architectures separately via a trace-driven simulation approach. Artemis [18] extends SPADE by involving virtual processors and bounded buffers. TAPES [19] abstracts functionalities by processing latencies covering the interaction of associated sub-functions on the architecture without actually running application code. MESH [20] treats resources, software and schedulers/protocols as three abstraction levels that are modelled by software threads.

The main contribution of this article is to present the seamless integration of distributed GENESYS application architecture modelling with ABSOLUT performance evaluation methodology to reduce the time and effort in the performance evaluation phase, enabling brisk architectural exploration. This is done via an extension of the distributed GENESYS application modelling style to form layered application architecture. Then the layers that are compatible to the workload model layers are identified and mapped to workload layers. Thus the resulting application model can be efficiently used as a starting point for performance evaluation reducing the time and effort.

III. MODELLING OF DISTRIBUTED GENESYS APPLICATIONS

In GENESYS, compliance of architectural views and concepts across application domains form the basis of cross-domain architectural style [21]. GENESYS reference architecture template provides core and optional services to application components. Core services are fundamental to any architecture [21]. Optional services, built on top of the core services, can be used in applications across multiple domains.

3.1. GENESYS Views

The modelling process starts by describing a set of views defined in GENESYS that are sufficient for the modelling objective. These views are instantiated via UML2.0 MARTE profile and illustrated in conjunction with an Office Security Application case study. GENESYS use case view describes the functionality of a system at a higher abstraction level by means of use cases. The syntactical view describes the syntax the servers understand in order to access their services. Sub-systems together with their interfaces (set of services) are conceived as servers that admit different messages from the application (client). The behavioural view reflects the behavioural aspects of an application and its encompassing services.

3.2. Non-Functional Properties

Non-functional properties from the end-user perspective are identified and elaborated in the syntactical view. Firstly they are shown in the extended behavioural view and later on validated by the performance simulation. We focus in the sequel on one non-functional property, FrameRate, showing the way it is carried through the design process. This is outlined in Figure 1.

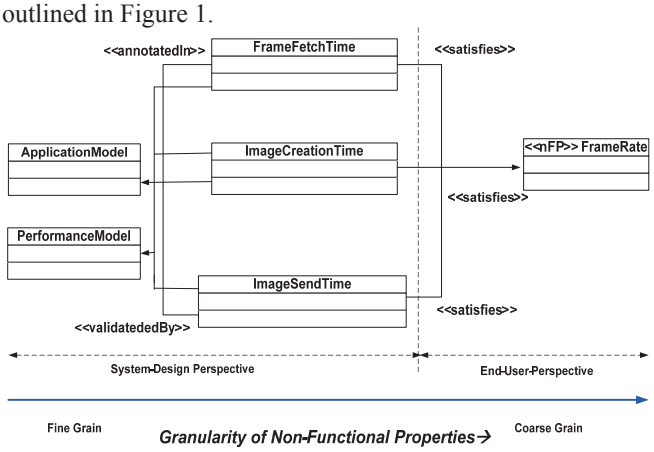

Figure 1: Carrying FrameRate through the application modelling and performance simulation process

3.3. Case study

The case study describes an Office Security application (previously described in [22] elaborate in this article for completeness) hosted on a mobile device owned by a member of security staff which communicates with three devices to avail different services. The PersonCounterSubSystem gives the number of occupants in office and a video of office entrance. The OfficeVideoSubSystem provides high resolution office video. The FaceTrackerSubSystem provides the information about number and video of occupants sitting on the bench. Each device communicates with its respective streaming device fitted with integrated camera mounted at an appropriate position in office and stream video frames to their client devices on demand as shown in application views.

1) Use case view

Use case view shows a system-level capability SelectTheSecurityService shown in Figure 2 in terms of device services.

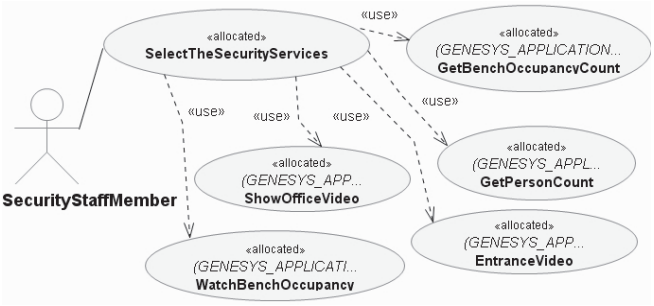

Figure 2: Use case view.

2) Behavioral View

Behavioral view shows the behavior of an application. The Application invokes different services as use case evolves as shown in Figure 3.

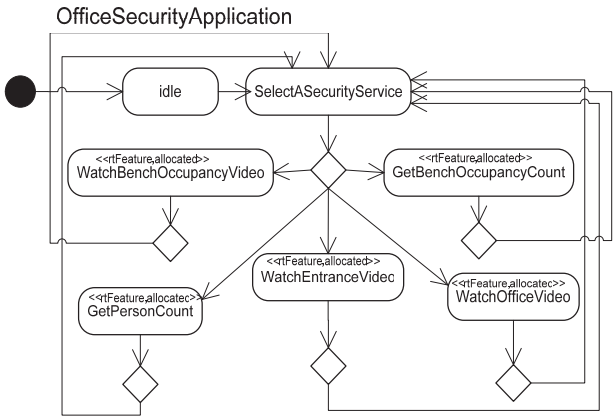

Figure 3: Operation of the Office Security Application.

3) Syntactical View

The syntactical view shows messages admitted by the sub-systems. Only three out of six subsystems that serve the application directly are shown in Figure 4. The used stereotypes and remaining three subsystems are described in detail in [11]. The non-functional properties are shown in Figure 5.

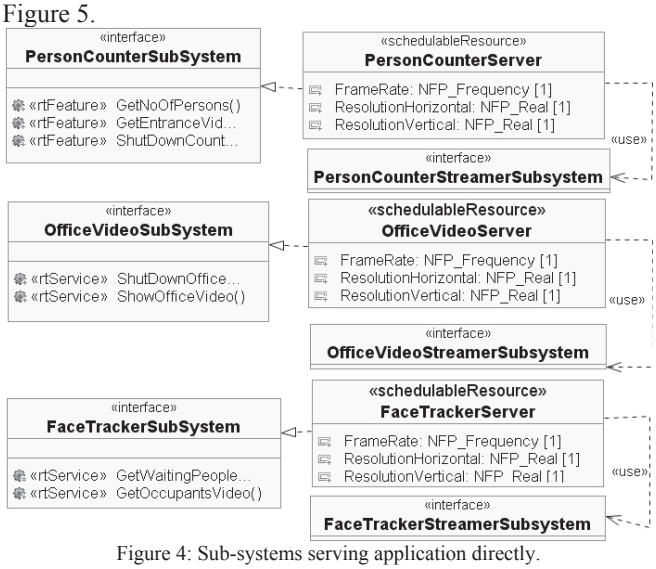

Figure 4: Sub-systems serving application directly.

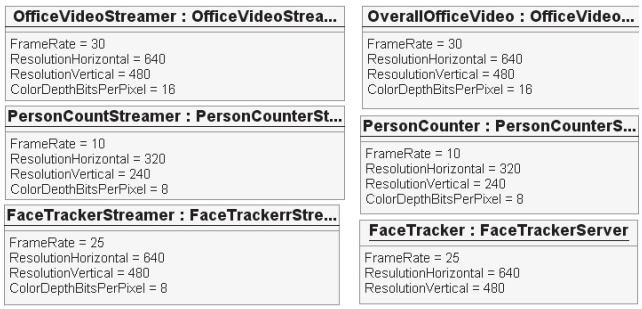

Figure 5: Non-functional properties.

IV. PERFORMANCE ANALYSIS APPROACH

The ABSOLUT performance evaluation approach follows the Y-chart model consisting of application workload and platform model [7]. After mapping the workloads to the platform, the models are combined for transaction-level performance simulation in SystemC [7].

4.1. Application Workload Model

The workloads consist of three layers [7] as shown in Figure 6.

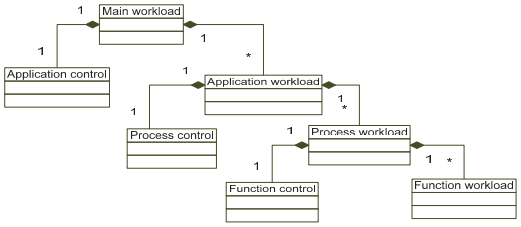

Figure 6: The application workload layers

4.2. Execution Platform Model

The platform model is an abstract hierarchical representation of actual platform architecture. It is composed of three layers [7] as shown in Figure 7. Each layer has its own services [7]. Services in subsystem and platform architecture layers are invoked by application workload models as explained in [7].

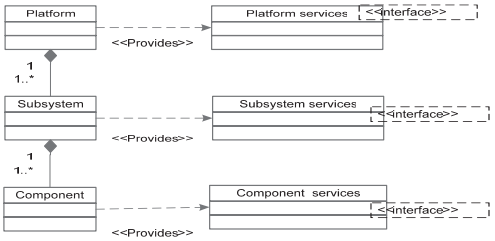

Figure 7: The platform architecture model layers.

V. PERFORMANCE MODELLING OF DISTRIBUTED GENESYS APPLICATIONS

For performance modeling of distributed GENESYS applications, the application model is extended to form a layered hierarchal architecture. Afterwards the corresponding layers in the ABSOLUT workload models are identified [8]. This reduces the time and effort in the performance evaluation phase as shown in Figure 8.

Performance evaluation of distributed applications via ABSOLUT demands the modeling and integration of Transport and MAC protocols, different modulation techniques, coding schemes and channel models. MAC and transport models have been integrated to ABSOLUT as described in [9]. The performance simulation of distributed GENESYS applications via ABSOLUT is summarized in Figure 8.

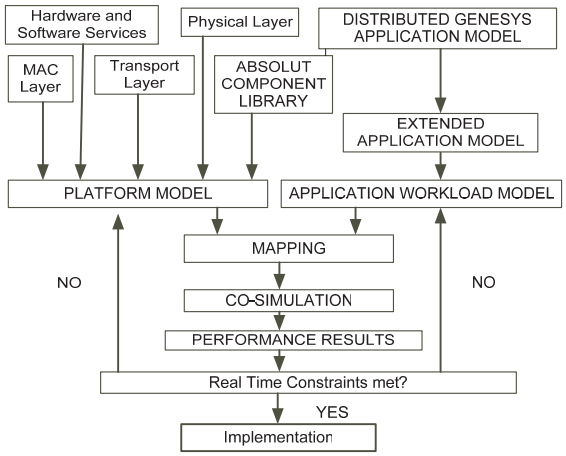

Figure 8: The performance simulation approach.

5.1 Integration of Transport, MAC and Physical Layers

Extension of ABSOLUT for the performance evaluation of distributed applications requires the modelling of protocols

operating at different layers of OSI model. This in turn demands a mechanism for instantiating new H.W and S.W services. These services are registered to the ABSOLUT operating system (OS) model and are used by the application workload models. Furthermore, the services operating at a higher layer of OSI model can use lower layer services for example transport-level services such as TCP can use Data-link level services such as IEEE 802.11 MAC protocols for the transmissions of frames of a packet as shown in

Figure 9. These services are instantiated by deriving them from the OS_Service base class as shown in

Figure 9 which implements the Generic_Serv_IF. The modelling and integration of highly accurate Datalink and Transport-Level services is explained in [9].

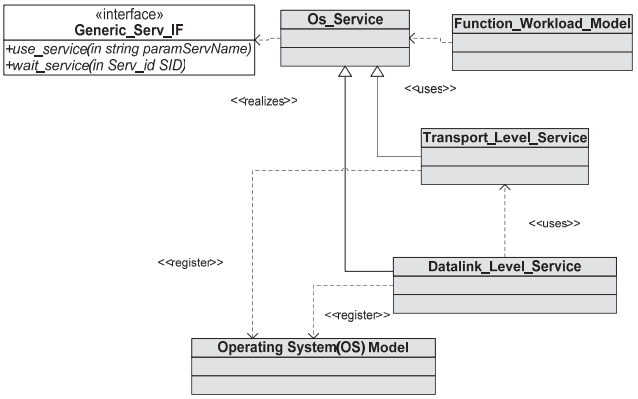

Figure 9: OS services implemented to model use cases spanning multiple devices and for modelling BSD API as OS services.

The OS_Service base class implements the functionality related to the scheduling of requests of process workload models (implements Generic_Serv_IF) for a particular service via priority queues and informs the requesting process workload model upon service completion. This is shown in Figure 10.

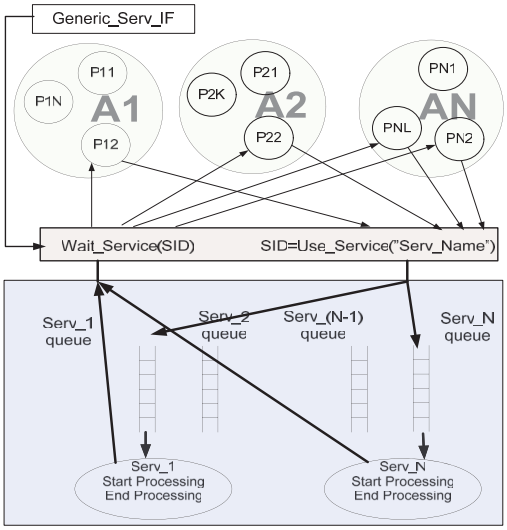

Figure 10: Diagram showing the mechanism employed by OS services to execute requests of processes.

The services derived from OS_Service base class merely implement the service specific functionality, making the process of modelling new services straight forward. The services operating at each layer of the OSI model are registered to the OS model via a unique service name [9]. A registered service is accessed by process models using its unique name assigned to the service while registering it to the OS model [9].

5.2 Extended GENESYS Behavioural View

To seamlessly integrate the distributed GENESYS application architecture modelling to ABSOLUT [8], the behavioural view of GENESYS application model is extended to extract a layered hierarchical structure of applications and the corresponding layers in the application workload models are identified. In this way, the application model acts as a blue print for the application workload models, reducing the time and effort in the performance evaluation phase.

In GENESYS [8] [21], a distributed use-case is presented by the behavioural view as a controlled collaboration of service providers and service requesters (both called Servers in GENESYS instead of clients and servers [8]) running on different devices. For example, if the use case involves the collaboration among "*k"* GENESYS Servers, we can write.

$E_a = \{ C_E, Serv_1, Serv_2, ..., Serv_k \}$,

where C_E represents the control mimicking the collaboration among nodes in order to satisfy the end-user use-case. The corresponding workload model layer is

$W = \{C_W, Servwld_1, , ..., Servwld_k\}$,

where $Servwld_i$ is workload model of a GENESYS server and C_W shows the control. Each of these workload models is an Application-Level ABSOLUT workload model lying at the application layer as shown in Figure 6. Each GENESYS Server contains a set of processes and control, .i.e., $Serv_i = \{C_P, P_1, P_2, ..., P_N\}$,

where *Pi* is the model of a single process in a GENESYS server. It should be noted that all processes of a single GENESYS server communicate via Inter Process Communication (IPC) and are scheduled by the same platform .i.e., operating system of the same device [25]. In other words, a single GENESYS server cannot span multiple devices. A GENESYS server might contain a single process. In that case there will be no control. The corresponding workload model layer is

$Servwld_i = \{C_{PM}, PM_1, PM_2, ..., PM_N\}$,

where *PMi* is the model of the *ith* process workload model of a GENESYS server workload model. Each Process can have function calls or platform service calls. The processes models of a single GENESYS communicate via ABSOLUT IPC models as elaborated in [25] and are scheduled by the operating system of the same platform model.

The processes of a GENESYS server can call library functions, system calls and functions of user-space code. The processes of GENESYS servers running on the same platform communicate via IPC [25]. The processes running on different platforms communicate via transport technologies for example TCP [9].

The corresponding Process level ABSOLUT workload models of a GENESYS server call function workload models of user space code (automatically obtained by ABSINTH [7]), function workload models of external library functions (obtained automatically by ABSINTH2[26]) and OS services modelled as OS_Services [9]. The process workload models of a GENESYS servers running on same platform communicate via ABSOLUT IPC ABSOLUT model [25]. The process level workload models running on different

platforms communicate via Transport-Level OS services [9]. The control and functionalities of the FaceTrackerStreamerServer (which consists of a single process) are shown Figure 11. The non-funcitonal property .i.e. FrameRate is elaborate in the application model element representing FaceTrackerStreamerServer [22]. This FrameRate is assigned a value of 25 Frames/sec as shown in Figure 5. This non-functional property is further refined to three non-functional properties from the design and implementation perspective .i.e., FrameRetrievalTimeMax, ImageCreationTimeMax, FaceDetectionTimeMax and ImageSendingTimeMax as shown in Figure 11. These refined non-functional properties are annotated in the behavioural view to their corresponding functionalities .i.e., Get a Frame, Detecet and Draw Faces, Create Image and Send the Image as shown in Figure 11. The OPENcv [27] library functions .i.e., cvQuerryFrame and cvCreateImage and user-space functions .i.e., DetectAndDearFaces and SendImage providing these functionalities are mentioned below the name of these functionalities. Each of these non-functional requirements are analysed in the performance simulation phase to check whether the required FrameRate has been achieved by the FaceTrackerStreamerServer. Due to the pipelined nature of the functionalities, each of them has to be performed within 1/25 seconds (to fulfil the required frame rate). The function SendImage is a wrapper around the TCP/IP BSD socket API's send() function [28].

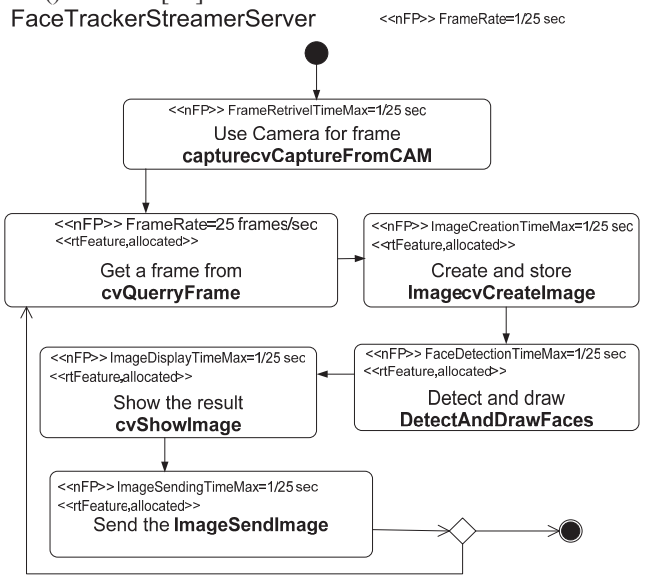

Figure 11: FaceTrackerStreamerServer control with functionalities mentioning refined non-functional properties

Hence a single process of an AN or SN *"P_i"* can be represented as $Pi=\{C_F,F_1,F_2,\ldots,F_N, S_1, S_2,.. S_K\}$, where Fi= a function and Si is a service requested from platform. The corresponding workload model layer is

$PMi=\{C_{FM},FM_1,FM_2,\ldots,FM_N, SM_1, SM_2,.. SM_K\}$,

where *FMi* is a function workload model and *SMi* is a platform service model(OS_Service).

5.3 Extracting Workload Layers

The mapping between the GENESYS Application model layers and the corresponding Workload model layers are elaborate in Table 1.

Table 1: Comparing GENESYS application model layers and ABSOLUT Workload layers.

Application Layers	Workload Model Layers
$E_a=\{C_E,Serv_1,\ldots,Serv_k\}$	$W=\{C_W,Servwld_1,\ldots,Servwld_k\}$
$Serv_i=\{C_P,P_1,P_2,\ldots,P_N\}$	$Servwld_i=\{C_{PM},PM_1,\ldots,PM_N\}$
$Pi=\{C_F,F_1,..,F_N,S_1,..,S_K\}$	$Mi=\{C_{FM},FM_1,..,FM_N,SM_1,..SM_K\}$

The word mapping does not mean the automatic extraction of workload models from the application models but refers to the identification of workload model elements corresponding to application model elements to facilitate application workload modeling.

VI. Performance Evaluation and Results

In the case of non-distributed applications, the overall performance model consists of a single platform model to which one or more applications workload models are mapped. These application models represent the processing load of the whole use-case [8]. In case of distributed applications, each server and client (both called Servers in GENESYS) in real use-case is modelled as a separate application-level workload model. Each Application-Level workload model of a GENESYS server instantiates a process workload model mimicking its execution in the real use-case. In case of performance models of distributed applications, at least two process workload models are hosted on different platform model instances. The performance results for all the platform models are obtained separately and analysed to perform optimizations if the non-functional properties are not satisfied.

6.1 ABSOLUT Performance Model

In the case study, each GENESYS server presented in the application model is mapped to a separate multi-core platform model to analyse the performance results and identify the potential bottlenecks at the software and hardware side. The overall performance simulation model is shown in Figure 12.

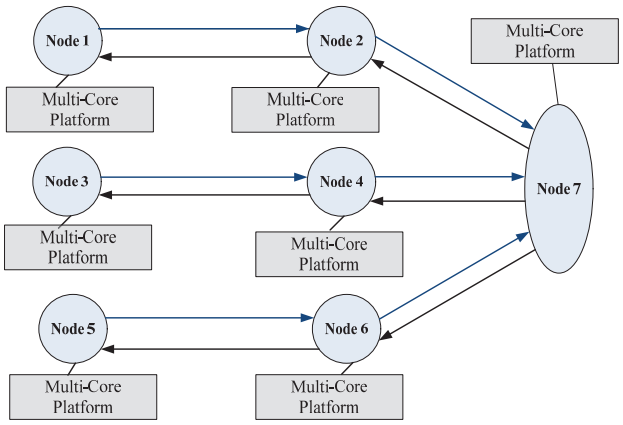

Figure 12: Performance model of Office Security application

Node 1 and Node 2 represent the PersonCounter and the PersonCounterStreamer server. Node 3 and Node 4 represent the OfficeVideo and OfficeVideoStreamer server whereas Node 5 and Node 6 represent the FaceTracker and FaceTrackerStreamer server. Node 7 represents the application which is in the form of a control [8].

6.2 ABSOLUT Platform Model

Each ABSOLUT platform model used in the case study consists of an ARM Cortex-A9 multi-core processor [29] model consisting of four cores along with SDRAM, a POWERVR SGX40 graphics accelerator and an Image signal

processor as shown in Figure 13. These component models are connected via an AMBA bus model.

In ABSOLUT methodology, the application models contain approximate timing information and the execution platform is modelled at transaction level [7].
The application workload models do not include accurate address information cache misses are modelled statistically [7] Processor performance is taken into account by defining clock frequency of cores. Architecture efficiency of cores is modelled as average cycles-per-instruction (CPI) value.

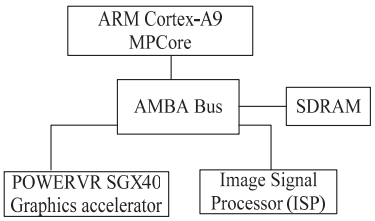

Figure 13: ABSOLUT platform model

Each core of ARM Cortex-A9 MP Core model has an L1 instruction and L1 data cache as shown in Figure 14**.**

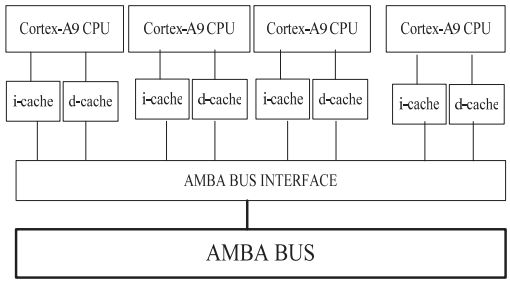

Figure 14: Diagram showing the quad-core processor (ARM Cortex-A9 multi-core processor) model used in the performance

6.3 Application workload Model

All the Servers elaborated in the application model were programmed using OpenCV library [27]. The tool used for the workload extraction is ABSINTH-2 [26] which automates the workload generation of external libraries.

6.4 Simulation Scenario

The simulations were carried out in WLAN environment and the simulation parameters for physical and MAC layer are adjusted accordingly as shown in Table 2. These parameters include IEEE 802.11 DCF configuration parameters and channel bit rate e.t.c., as explained in [9].

Table 2: Experiment parameters

Parameters	**Values**
SIFS	10 micro seconds
DIFS	50 micro seconds
Slot Interval	20 micro seconds
Preamble Length	144 bits
PLCP header Length	48 bits
Channel bit rate	2 Mbps
CWmin	32
CWmax	2048
CWo	32
EW	16

6.5 Co-Simulation and performance results

During the execution of application, the security staff member requests the bench occupancy and the video of occupants. The video frames are streamed form the PersonCounterServer to the mobile device of the security staff member. Then the staff member invokes other Servers one by one, switching between them after 1→2 minutes each, the Servers then invoke the corresponding SteramerServers to provide the required services to the application.

Each GENESYS server workload model is mapped to its respective platform as shown in Figure 12 and the resultant performance model is run to obtain performance results. The results of all the platforms and their hosted GENESYS servers are written to one text file in the form of different sections, one for each platform and its hosted GENESYS servers. We only present the performance results of the platform hosting the FaceTrackerStreamerServer.

The simulation execution can be easily exited after any pre-decided simulation time, for example after 20 seconds (time in terms of SystemC time model) or another event in the simulation for example the number of streamed packets from one StreamerServer to corresponding Server or from a Server to the Application. When the pre-decided condition is met during simulation, the sc_stop() function is called. After that the destructor of the results reporting class is called which writes the gathered simulation results to a text file for analysis.

1) Performance Results (Platform)

Since the FaceTrackerStreamerServer was implemented entirely as software, the Graphics Accelerator and Image Processor services available from the platform were not used. Therefore only the utilization of the processor cores of platform hosting FaceTrackerStreamerServer is shown in Figure 15. The simulation was run for streaming of 10, 100 and 1000 packets. The solid bar corresponds to 10 packets, bar with horizontal pattern shows use-case of 100 packets and diagonal pattern corresponds to 1000 packets.

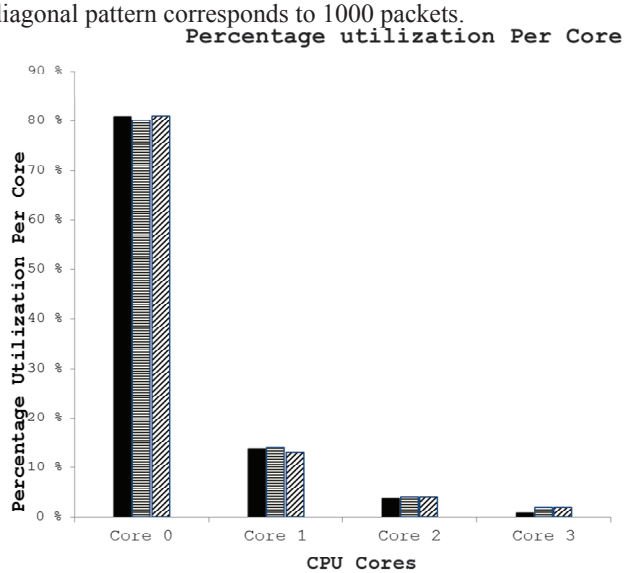

Figure 15: Utilization time of processor cores as compared to overall Utilization time of the CPU

The cache statistics of the platform hosting FaceTrackerStreamerServer are shown below for 1000 video frame transmissions.

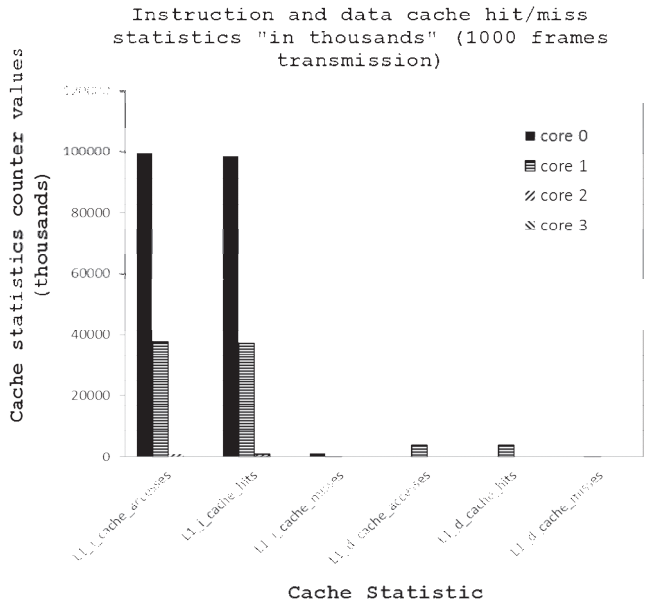

Figure 16: Cache hits miss statistics of the platform hosting FaceTrackerStreamerServer Server

The performance statistics related to the platform services (MAC and Transport protocol models) are recorded via probes. The performance statistics of Transport (UDP) and MAC (IEEE 802.11 DCF) level protocols models are shown in Table 3.

Table 3: MAC and Transport Performance statistics

MAC/Transport Performance statistic	Values
Throughput at MAC-Level	.99
Throughput at Transport-Level	.98
Average Frame Delays	.48 millisec
Average Transport Delays	2.1 millisec
Frame loss rate (Percent)	.022%
Packet Loss rate(Percent)	.983%

The results in Table 3 will only satisfy the non-functional property (FarmeRate), if all the functions in the FacerTrackerStreamerServer which make use of these OS_Services satisfy the non-functional properties from the design perspective.

In case of FacerTrackerStreamerServer, only the SendImage function Figure 11 makes use of Transport and MAC-Level ABSOLUT protocols models (OS_Services). The function SendImage is a wrapper around the TCP/IP BSD socket API's send() function which is modelled as a Transport level OS_Service. As shown in Figure 11, SendImage function must be executed within 40 milliseconds (1/25 seconds) in order to satisfy the required FrameRate of 25 Frames/Second. The processing time of this function along with the other FacerTrackerStreamerServer functions are presented next.

2) Performance Results (Application). Validating Non-Functional Properties

By analysing the processing times and percentage utilization of multi-core processor by the user-space code and external library functions, we can find the potential bottlenecks in the application implementation which will help to perform required optimizations. In other words, after identifying the functionalities which can affect a particular non-functional property, the processing times of these functionalities are analysed to find out whether the implementation of these software components satisfies this non-functional property.

We now elaborate the way the non-functional property FrameRate is analysed and validated by the performance simulation results. This non-functional property is annotated in the application syntactical view and refined to three non-functional properties in the extended behavioural view as shown in Figure 11. Due to the pipelined nature of the execution of these functionalities, each must be executed within 1/25 seconds (40 milliseconds) in order to achieve a frame rate of 25 frames/seconds. The processing times and the percentage processor utilization of the aforementioned functions are shown in Figure 17 and Figure 18**.** It is seen that all the operations are performed within 22000 microseconds or 22 milliseconds, therefore ensuring that the current implementation of FaceTrackerStreamerServer will satisfy the required frame rate when run on the platform shown in Figure 13. The results show that the SendImage function takes less than 5 milliseconds which is well below 40 milliseconds required to achieve the required FrameRate. In this way, the results presented in Table 3 are also validated since it is the only function that makes use of a Transport level OS_Service. In other words the performance of MAC and Transport protocols is sufficient to satisfy the use case.

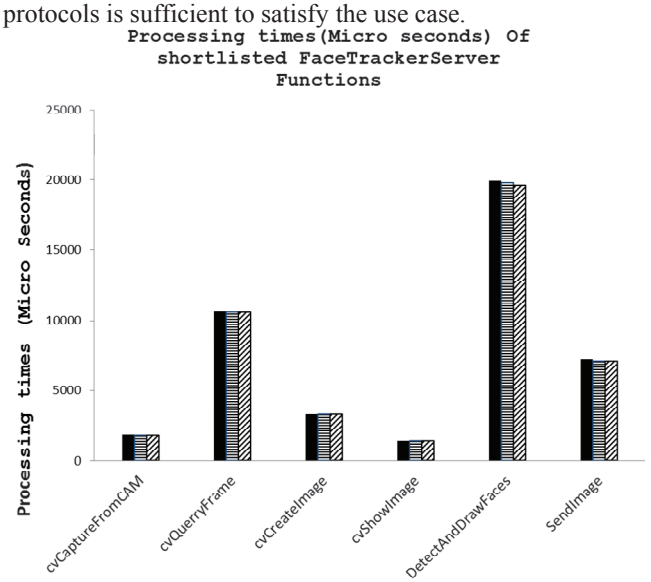

Figure 17: Execution times of functionalities attributing to Frame rate in Face Tracker Subsystem (platform hosting FaceTrackerStreamerServer)

The processor utilization graph shows that drawing and detection of faces takes 41% of the multi-core processor time in proportion to the overall time taken by all the FaceTrackerStreamerServer functions.

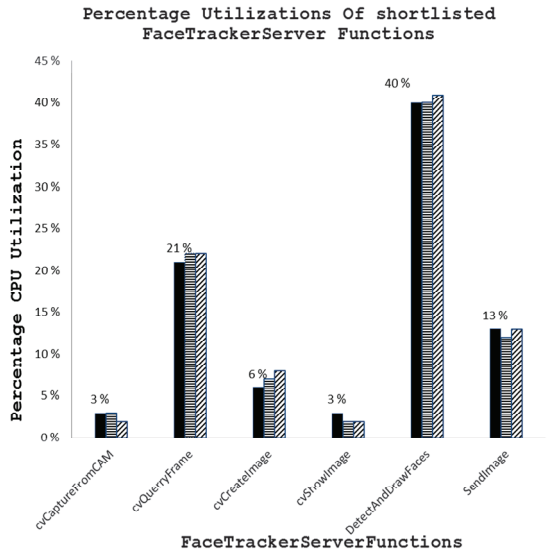

Figure 18: Execution times of functionalities attributing to Frame rate in Face Tracker Streamer Subsystem (platform hosting FaceTrackerStreamerServer)

The obtained performance results are used to perform appropriate changes in the application models by replacing the software components with more light weight implementations or by making changes in the platform model if the performance requirements (non-functional properties) are not met. If the performance requirements are met by all the platform and software components, the architectural exploration stops and the implementation phase starts.

VII. CONCLUSION AND FUTURE WORK

The distributed GENESYS application architecture modelling methodology was extended for linking it with the ABSOLUT workload modelling approach. The behavioural view of the application architecture was extended to obtain a layered application model that can be mapped to the workload model layers used in the performance simulation approach. The MAC and Transport protocols were modelled as highly accurate OS_Services which were used by the Process level workload models for communication with processes running on different devices (ABSOLUT platform models). The approach was experimented with an Office Security Application case study.UML2.0 MARTE profile was used as the modelling language and Papyrus UML2.0 as the modelling tool.

ACKNOWLEDGEMENTS

This work was performed the Artemis SOFIA project partially funded by TEKES and the European Union.

REFERENCES

[1] T. Noergaard. *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers.* ELSEVIER, UK; 2005, 640 p.

[2] K. Keutzer, A. Newton, J. Rabaey and A. Sangiovanni-Vincentelli,"System-level design: orthogonalization of concerns and platformbased design", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 19 (12), 2000, pp. 1523 - 1543.

[3] A. Sangiovanni-Vincentelli. Defining Platform-Based Design. www.eedesign.com/story/OEG20020204S0062, EEDesign. February 2002.

[4] J. Arlow, Ila Neustadt. *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design.* 2nd Edition. Addison-Wesley, USA; 2008, 592 p.

[5] B. Kienhuis, E. Deprettere, K. Vissers and P. van der Wolf. Approach for quantitative analysis of application-specific dataflow architectures. The IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP '97), pp. 338–349, Zurich, Switzerland. July 1997.

[6] http://www.genesys-platform.eu

[7] Jari Kreku, Mika Hoppari, Tuomo Kestila, Yang Qu, Juha-Pekka Soininen and Kari Tiensyrja, Combining UML2 and SystemC Application Platform Modelling for Performance Evaluation of Real-Time Systems, EURASIP Journal on Embedded Systems, volume 2008, ARTICLE ID 712329.

[8] Subayal Khan, Susanna Pantsar-Syväniemi, Jari Kreku, Kari Tiensyrjä and Juha-Pekka Soininen, "Linking GENESYS Application Architecture Modelling with Platform Performance Simulation", in Proceedings of the 12th Forum on Specification and Design Languages (FDL 2009), September 22-24, Sophia Antipolis, France, 2009.

[9] Subayal Khan, Jukka Saastamoinen, Mikko Majanen, Jyrki Huusko and Jari Nurmi. Analyzing Transport and MAC Layer in System-Level Performance Simulation, International Symposium on System-on-Chip 2011,Tampere, Finland, October 31 - November 2, 2011.

[10] Lappeteläinen, Antti et. al., 'Networked Systems, Services, and Information - The Ultimate Digital Convergence'. 1st International Conference on Network on Terminal Architecture, June 11, 2008, Helsinki.

[11] Instantiation and feasibility evaluation of NoTA SOAD via MARTE profile and binary instrumentation. Khan, Subayal; Tiensyrjä, Kari; Nurmi, Jari. The 7th International Conference and Expo on Emerging Technologies for a Smarter World. CEWIT 2010. Incheon, 27 - 29 Sep. 2010.

[12] L. Bass, P. Clements and R. Kazman. Software Architecture in Practice. Addison-Wesley, 1999, 528 p.

[13] http://www.uml.org

[14] USC Center for Software Engineering, Guidelines for Model-Based (System) Architecting and Software Engineering, http://sunset.usc.edu/research/MBASE, 2003.

[15] D. Garlan, R. T. Monroe and D. Wile. Acme: An Architecture Description Interchange Language. Proceedings of CASCON '97, November 1997.

[16] R. Roshande, B. Schmerl, N. Medvidovic and D. Garlan, D. Zhang. Understanding tradeoffs among different architectural modeling approaches. Software Architecture, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference on software architecture.12-15 June 2004, pp. 47 – 56

[17] P. Lieverse, P. van der Wolf, K. Vissers, and E. Deprettere, A methodology for architecture exploration of heterogeneous signal processing systems. Kluwer Journal of VLSI Signal Processing 29 (3), 2001, pp. 197-207.

[18] A. Pimentel and C. Erbas. A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels. IEEE Transactions on Computers, vol. 55, no. 2, Feb. 2006, pp.99 –112.

[19] T. Wild, A. Herkersdorf and G.-Y. Lee. TAPES—Trace-based architecture performance evaluation with SystemC. Design Automation for Embedded Systems, Vol. 10, Numbers 2-3, Special Issue on SystemC-based System Modeling, Verification and Synthesis, 2006, pp 157-179.

[20] J.M. Paul, D.E. Thomas, and A.S. Cassidy. High-Level Modeling and Simulation of Single-Chip Programmable Heterogeneous Multiprocessors. ACM Transactions on Design Automation of Electronic Systems, Vol. 10, No. 3, 2005, pp. 431-461.

[21] E. Ovaska, A. Balogh, S. Campos, A. Noguero, A. Pataricza, K. Tiensyrjä and J. Vicedo. *Model and Quality Driven Embedded Systems Engineering* VTT, Espoo, 2009, 208 p.

[22] Subayal Khan, Kari Tiensyrjä, "Instantiating GENESYS Application Architecture Modeling via UML 2.0 constructs and MARTE Profile", in Proceedings of 13th Euromicro Conference on Digital System Design (2010), September 1-3, Lille, France, 2010.

[23] http://www.isi.edu/nsnam/ns/

[24] http://www.omnetpp.org/

[25] Multi-threading support for system-level performance simulation of multi-core architectures. ARCS 2011 - 24th International Conference on Architecture of Computing Systems.Como, Italy. 02/22/2011 - 02/23/2011. Saastamoinen, Jukka; Khan, Subayal; Tiensyrjä, Kari.

[26] Application workload model generation methodologies for system-level design exploration. DASIP 2011. November 2-4 2011.Tampere, Finland. Jukka Saastamoinen, Jari Kreku.

[27] http://opencv.willowgarage.com/wiki/

[28] http://en.wikipedia.org/wiki/Berkeley_sockets

[29] http://www.ti.com/omap

PUBLICATION VII

Performance Evaluation of distributed NoTA applications on multi-core platforms

Suabyal Khan, Jukka Saastamoinen, Jyrki Huusko
VTT Technical research Center of Finland,
FI-90570, Oulu, Finland
e-mail:{subayal.khan,jukka.saastamoinen,
jyrki.huusko }@vtt.fi

Jari Nurmi
Tampere University of Technology,
Department of Computer Systems
P.O. Box 553, Korkeakoulunkatu 1,
FIN-33101 Tampere, FINLAND
jari.nurmi@tut.fi

Abstract

For future applications running on nomadic devices operating in smart spaces, the availability of the services and the quest for better alternative services will go hand in hand to enable the best-possible experience for the end-user. To realize it, a service-level interoperability solution must be devised that would enable applications to access services over heterogeneous platforms and various transport technologies. The recent extension of the Network-on-Terminal-Architecture (NoTA) supports service-level interoperability between mobile devices via a device interconnect protocol (NoTA DIP), enabling applications to access and discover services over multiple transport technologies in a seamless manner. A brisk performance evaluation phase is required for evaluating the feasibility of new NoTA applications on modern multi-core based mobile device platforms. To achieve this goal, NoTA Service Oriented Architecture (SOA) application components .i.e., Application Nodes (ANs) and Service Nodes (ANs) are first refined to encompassing processes and then to platform services and functions to form a layered application architecture. In next step, the NoTA SOA application model layers are mapped to ABSOLUT workload model layers employed in performance simulation. Furthermore, the ABSOLUT workload models corresponding to different NoTA DIP implementations for example NoTA DIP kernel implementation and NOTA DIP Daemon mode must be modeled and integrated to ABSOLUT. The approach is experimented with a case study. The modeled components and approach is not limited to NoTA SOA and can be used for the performance evaluation of other distributed service oriented application architectures. MARTE UML2.0 profile, Papyrus UML 2.0 modeling tool and SystemC were used for modeling and simulation.

Key Words: NoTA, SystemC, Performance Simulation.

I. INTRODUCTION

In order to provide the user with a more rich experience in pervasive environment, the applications running in modern multi-core platform devices needs to take advantage of the available services hosted by other devices in the same environment. This demands both service accesses over various transport technologies and common service discovery mechanisms.

The NoTA SOA [1] implements this mechanism in the form of a device interconnect protocol (DIP) that abstracts away the complexity and algorithmic details involved in service discovery and access over multiple transports and provides a simple modified Socket API interface to application programmer collectively called NoTA BSD-SOCKET API functions [1].

Deployment of NoTA applications is challenging not only due to the heterogeneous parallelism in the multi-sore mobile platforms, but also due to performance and energy constraints. For efficient product development, it is of pivotal importance that the NoTA application design phase acts as a blue print for the system-level performance evaluation. For performance simulation Abstract workload based performance simulation (ABSOLUT) is used which employs model-based approach for both application and platform [2].

So far ABSOLUT has not been used to evaluate the use cases that span multiple devices operating in a ubiquitous environment. Performance evaluation of distribute NoTA applications via ABSOLUT demands the design, implementation and integration of physical layer models for example channel models, modulation and coding techniques, and transport layer models for example UDP and TCP and medium access control (MAC) protocols for example IEEE 802.11 DCF. Highly accurate physical, MAC and transport layer models have been integrated to the ABSOLUT framework [3]. Nevertheless, NoTA SOA employs device interconnect protocol for service-level interoperability, .i.e., service discovery and access. NoTA is implemented as an external library and as platform service. When used as an external library, it operates in two modes .i.e., single process (SP) and daemon mode [4]. When implemented as a platform service (Operating system (OS) service), it can be used by the application in the same way as other platform services for example Berkeley BSD Socket API functions.

In single processor mode NoTA is compiled with the application as a static library. In daemon mode it runs in the background and serves the applications. In third case it operates as a platform service implemented as a part of the OS kernel.

The workload models corresponding to the three NoTA modes have to be designed and integrated to ABSOLUT. Furthermore, for the seamless integration of application design and performance simulation the NoTA application model behavioural view is extended to form a layered hierarchical structure. The ABSOLUT workload model layers corresponding to these layers are identified. This mapping between the application model and workload model layers allows seamless integration of application design and performance simulation. This back-to-back application design and performance simulation reduces the time and effort in the performance simulation phase and reduces the time to market by brisk iterations in the architectural exploration phase [6].

The rest of the paper is organized as follows: Section II gives a brief outline of landmark performance simulation techniques. Section II gives an overview of NoTA SOA and

elaborates the modelling of NoTA applications via a case study. Section IV elaborates the ABSOLUT performance simulation approach and Section V shows extensions of ABSOLUT and NoTA Device Interconnect protocol (DIP) Workload modelling and the integration of these models to ABSOLUT.

The Section VI elaborates the overall performance model and shows the simulation results. Conclusions and future work are outlined in Section VII followed by acknowledgements and list of references.

II. Related work

Object Management Group (OMG) defines Model Driven Application Architecture (MDA) relying on efficient use of system models to facilitate transformations between different model types. Various Architectural Description Languages (ADLs) have been proposed. MBASE provides integrated models for capturing the product success, process and properties [7]. Acme relies on a core ontology comprising of seven elements representing architectural elements [8]. Mae [9] triggers the modelling, analysis, and management of different versions of architectural artefacts supporting domain-specific extensions to capture other system properties.

Performance modelling has been approached in different ways. SPADE [10] treats applications and architectures separately via a trace-driven simulation approach. Artemis [11] extends SPADE by involving virtual processors and bounded buffers. TAPES [12] abstracts functionalities by processing latencies covering the interaction of associated sub-functions on the architecture without actually running application code.

The main contribution of this paper is to elaborate the modelling and integration of NoTA DIP stack workload models to ABOLUT and link the NoTA SOA application modelling style [13] to the performance evaluation phase [2]. The latter is achieved via an extension of the NoTA SOA application modelling style to form layered application architecture. Afterwards, the corresponding layers in the ABSOLUT application workload model are identified. The mapping between the application and workload models results in a seamless integration of the application design and performance simulation phases. In other words, the resulting application model can be efficiently used as a starting point for performance evaluation reducing the time and effort involved in the performance simulation phase.

III. NoTA Application modeling

NoTA is a novel SOA which consists of three types of logical elements: Service Nodes (SNs), Application Nodes (ANs) and Device Interconnect Protocol (DIP). Service nodes are services that can be used by ANs and other SNs. Application nodes are the application functionalities composed of service calls and other logic. Communication between the Application and Service Nodes takes place always over the DIP. The DIP defines both socket based communication i.e., it supports both message and streaming type of data flows. NoTA DIP is divided into two main functional blocks. The first one is called H_IN which manages service registration, discovery, access and security. The second is called L_IN which is responsible for connecting the subsystems together. From a software architect's perspective, the applications supported by NoTA systems are modelled as NoTA SOA [13]. In other words, a NoTA application consists of a set of Application Nodes (ANs) and Service Nodes (SNs) which collaborate via NoTA Device interconnect protocol to satisfy a use case [1]. For modelling a novel SOA for embedded nomadic devices (in this case NOTA SOAD), UML 2.0 MARTE profile comes as a natural choice [13].

A. NoTA application views

The NoTA application modeling process starts by describing a set of views that are sufficient for the modeling objective. These views are instantiated by using UML2.0 MARTE profile and will be illustrated in conjunction with the RM Application case study. The use case view describes the functionality of a system at a higher abstraction level by means of use cases. The structural view defines the interface between an application and the sub-systems of the execution platform. The interfaces are implemented by the ANs and SNs. The syntactical view describes the syntax of the messages passed between ANs and SNs. The behavioral view reflects the behavioral aspects of an application and its encompassing services.

B. Non-Functional Properties

Non-functional properties from the end-user perspective are identified and elaborated in the syntactical view. Firstly they are shown in the extended behavioural view and later on validated by the performance simulation. We focus in the sequel on one non-functional property, FrameRate, showing the way it is carried through the design process for the design of a certain SN in the distributed NoTA application. This is outlined in Figure 1.

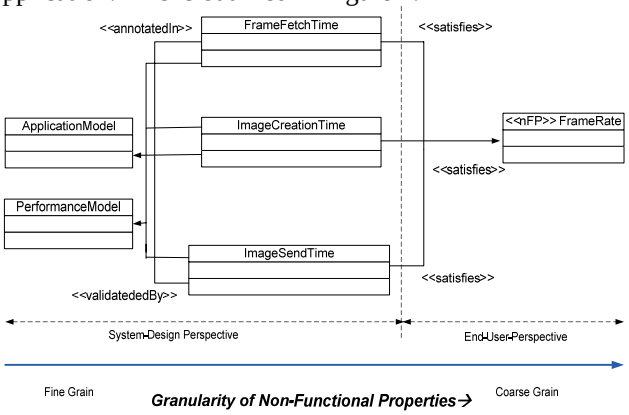

Figure 1: Carrying FrameRate through the application modelling and performance simulation process

C. Case Study

We now describe the modeling of Restaurant Multimedia (RM) Application. This application allows a customer to request his preferred multimedia service and is hosted on his mobile device. The application is like a control which requests any of the three Application Nodes (ANs) for viewing a News Channel, A music video or a movie from the available lists. Each of these ANs then requests its corresponding Service Node (SN) to access the streaming video. These nodes are their required and provided interfaces are clearly elaborated in the application model.

1) Application use-case view

The use case view shows a system level capability .i.e., selection of Multimedia Service, as shown in Figure 2.

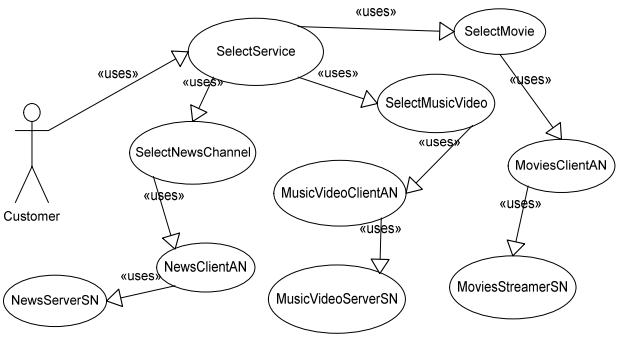

Figure 2: Restaurant Multimedia application use-case view

2) Application syntactical view

The syntactical view describes the syntax of messages passed between the ANs and application and also between ANs and SNs. This is shown in Figure 3.

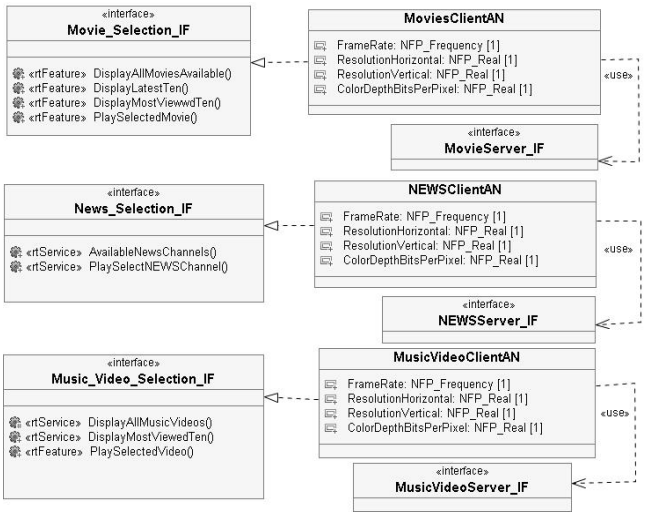

Figure 3: Diagram showing Interfaces realized and required by ANs.

The interfaces needed by ANs are provided by SNs and are shown similarly. The non-functional properties (NFPs) [13] are assigned values in respective slots of their instances and are shown in Figure 4.

Figure 4: The non-functional properties represented as slot values

3) Application Behavioral View

The behavioral view shows behavior of the application as shown in Figure 5. The functionalities in the diagram are allocated (implemented by) to ANs or SNs. These ANs and SNs must satisfy the non-functional properties annotated in the syntactical view for a better end-user experience. These non-functional properties are refined to a set of non-functional properties from the implementation perspective as shown in Section 5 and are validated by the performance simulation phase as shown in Section 6.

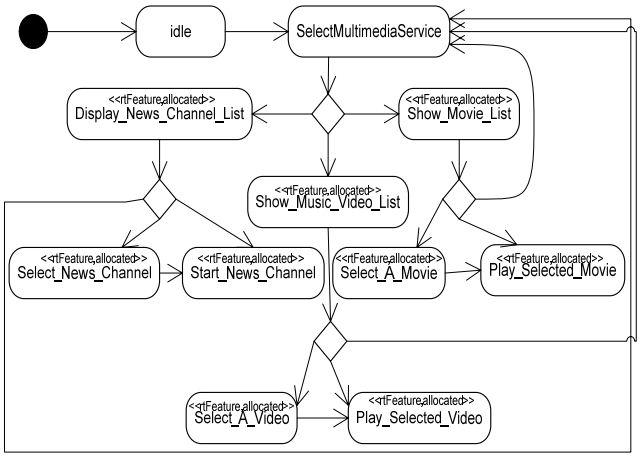

Figure 5: Behavioural view of Application.

IV. Performance Analysis Approach

The performance evaluation approach follows the Y-chart model consisting of application workload and platform model [2]. After mapping the workloads to the platform, the models are combined for transaction-level performance simulation in SystemC [2]. Based on the simulation results, we can analyze e.g. processor utilization, memory traffic and execution time.

The approach enables performance evaluation early, exhibits light modelling effort, allows fast exploration iteration, reuses application and platform models, and provides performance results that are accurate enough for system-level exploration [2].

The workloads consist of three layers .i.e., main workload, application workload and function workload [2]. The platform model is composed of three layers: component layer, subsystem layer, and platform architecture layer [2]. Each layer has its own services which are invoked by application workload models.

V. Performance Modeling of Distributed NoTA Applications

For performance modeling of distributed applications, workload models are created as described in [2]. Performance evaluation of distributed NoTA applications via ABSOLUT demands the modeling and integration of NoTA BSD API function models, transport and MAC protocols, different modulation techniques, coding schemes and channel models. Highly accurate Physical layer (channel coding, channel models and modulation techniques), MAC layer and transport layer models have been integrated to ABSOLUT as described in [3].The physical layer models have been modeled via itpp library [3]which are used by transport and MAC-layer models. The MAC and transport layer models were compared to the corresponding models of widely network simulators .i.e., ns-2 and OMNeT++ [14][15]. The results were 75-85% accurate as compared to these benchmarks and were always pessimistic. In other words, if the use-case requirements are

validated by the ABSOLUT MAC and transport models, the results are surely validated by ns-2 and OMNeT++ simulators since ABSOLUT models always give higher values of MAC and transport level delays and throughput under the same network conditions for example number of nodes and channel bit rate [3].

The new models integrated into the ABSOLUT framework include NoTA DIP workload models for daemon and NoTA OS Services (NoTA API functions available as OS services to applications). Also the NoTA application model has been extended for easy extraction of the application workload models. The performance simulation of NoTA SOA via ABSOLUT is summarized in Figure 6. The blue color indicates the novel contributions presented in this article. The workload models for external libraries are generated by modifying callgrind_annotate_ tool (a tool that presents the output of callgrind [16], call-graph generating cache and branch prediction prifiler [16]) which produces profiling reports that can be post-processed. The post-processing is done via a new tool called SAKE (abStract external library workload Extractor) [17], which generates the workload models for external libraries by post-processing the profiling output of the modified callgrind_annotate_tool [17].

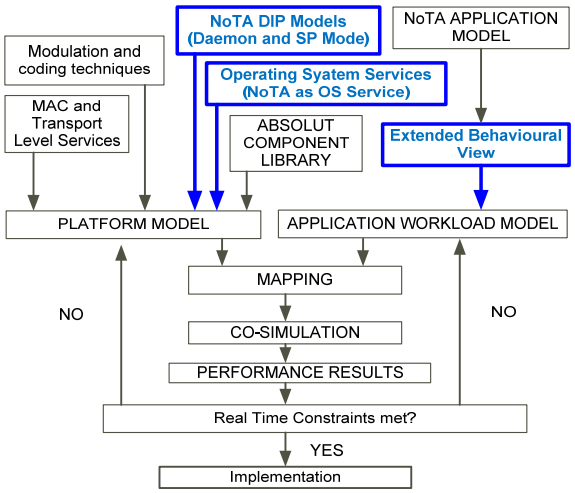

Figure 6: The performance simulation approach.

A. *Modeling NoTA DIP Workload Models*

NoTA DIP is available as an external library and has also been implemented as platform service implemented in LINUX Kernel [4]. When used as an external library, NoTA DIP operates in two modes .i.e., Single Process (SP) mode or Daemon mode [4]. In both cases, NoTA DIP services be requested by applications as modified NoTA BSD API functions. Linking NoTA Application architecture design to ABSOLUT demands the modelling of both NoTA implementations.

Modelling of NoTA (when modelling reflects the implementation of NoTA as platform services) workload models demands the implementation and integration of a novel mechanism in ABSOLUT for instantiating new H.W and S.W services. In this way the services required to extend ABSOLUT to NoTA such as IEEE 802.11 contention resolution schemes, NOTA API functions, message transmission and synchronization are implemented and integrated to ABSOLUT. This is shown in Figure 7.

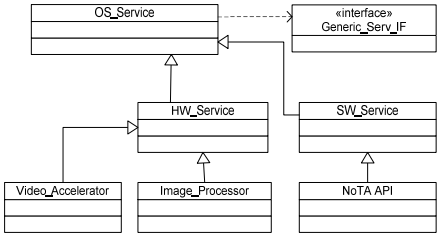

Figure 7. OS services implemented to model use cases spanning multiple devices and for modelling NoTA API as OS services.

The OS_Service base class implements the functionality related to scheduling the requests of processes via priority queues and informs the requesting process on service completion after taking it to running state again. This is shown in Figure 8.

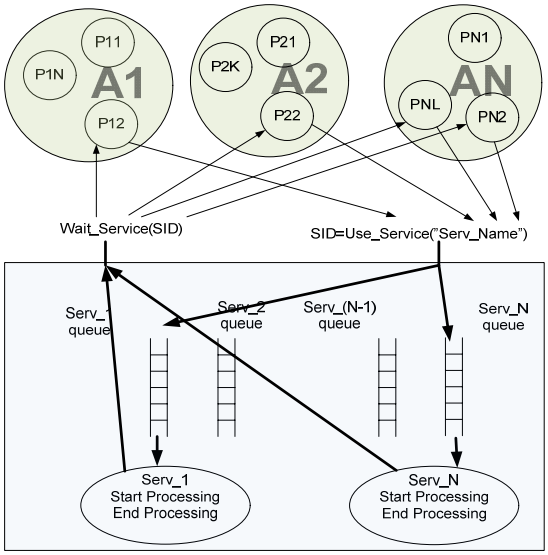

Figure 8: Processing of service requests by OS_Services base class

The derived services merely implement the functionality making the process of modelling new services straight forward. NoTA is available as an external library. ABSINTH2[17] automatically generates the workload models of NoTA API functions which are modelled as OS_Services. These services are registered to the OS model and executed in during the simulation when Process or Application level workload models request them from the OS. Other hardware and software services can also be derived from the OS_Service base class.

When NoTA is used as an external library, it can operate in two modes, .i.e., SP mode and Daemon mode. In case of SP mode, the workload models of NoTA modified BSD API functions are extracted via ANBSINTH2 [17] and sued by Process-Level workload models in the similar way as normal function workload models ANBSINTH2 [17].

In case of daemon mode, NoTA runs as a process in the background and serves the requests of other running processes via the same API (modified NoTA BSD API). The workload models of these API functions are extracted via ABSINTH2 [17]. Nevertheless, the requesting applications cannot execute these function workloads directly. These workload models are executed by the ABSOLUT Daemon models on receiving the corresponding requests (NoTA BSD API function calls). A generic

mechanism has been implemented which allows the modelling of daemon workload models that execute the requests of different processes. This is important since NoTA can also operate in daemon mode and complex use-cases could also involve daemons processing the request of other processes. The Daemon base class (Daemon_Workload) schedules the requests in a similar way as the OS_Service base class but is also a process in itself in the running state. This is shown in Figure 9. The NoTA daemon process workload model executes the BSP API function workload models extracted by ABSINTH2 [17] when requested by a process workload model of the AN or SN application model. Other daemons serving the process workload models can also be derived from the Daemon_Workload base class.

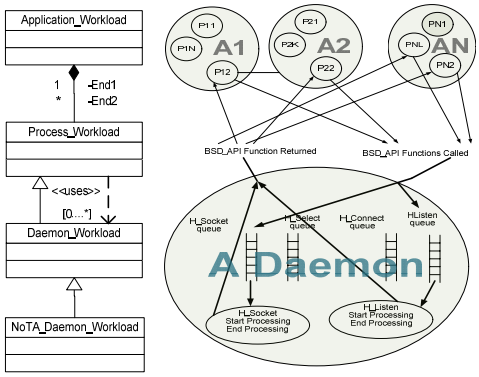

Figure 9: Relationship between daemon workload and process workload (left) and the way it serves the requests of processes.

B. *Extended NoTA Behavioural view*

In order to integrate the NoTA SOA to ABSOLUT [6] seamlessly, the behavioural view of NoTA application model is extended to extract a layered hierarchical structure of applications .The corresponding layers in the application workload models are identified. In this way, the application model acts as a blue print for the application workload models, reducing the time and effort in the performance evaluation phase.

In case of NoTA application model, the behavioural view represents a use_case as a controlled collaboration of ANs or SNs. Therefore,

$USE\_CASE = \{C, N_1, N_2, N_3, \ldots, N_N\}$,

where N_i is an *AN* or *SN* and *C* represents the control of the application shown in Figure 5.Corresponding workload model layer is

$USE\_CASE\_MODEL = \{C_{WLD}, NM_1, \ldots\ldots, NM_N\}$,

where NM_i is workload model of an AN or SN and C_{WLD} shows the control. Each of these workload models is an Application-Level ABSOLUT workload model lying at the application layer as shown in Figure 2.

Each AN or SN contains a set of processes and control, .i.e., $N_i = \{C_P, P_1, P_2, \ldots, P_N\}$, where P_i is the model of a single process in an *AN* or *SN*. It should be noted that all processes of a single *AN* or *SN* communicate via Inter Process Communication (IPC) and are scheduled by the same operating system of the same device. In other words, a single *AN* or *SN* cannot span multiple devices. An *AN* or *SN* might contain a single process. In that case there will be no control. The corresponding workload model layer is

$NMi = \{C_{PM}, PM_1, PM_2, \ldots\ldots, PM_N\}$,

where PM_i is the model of the *ith* process model. Each Process could either have function calls or service calls from the platform. The processes models of a single *AN or SN* communicate via ABSOLUT IPC models as elaborated in [18] and are scheduled by the operating system of the same platform model.

The processes models of an *AN* or *SN* can communicate with the process models of an *AN* or *SN* running on a different platform via transport layer services registered to ABSOLUT platform models. Highly accurate transport layer services have been modelled and integrated to the ABSOLUT framework as explained in [3].

The processes of an *AN* or *SN* can call library functions, system calls and functions of user-space code. For communication with other processes, they can call BSD_API functions or make use of IPC. The corresponding Process workload models call Function workload models automatically obtained by ABSINTH [2] and workload models for external library functions obtained by ABSINTH2[17]. The BSD API functions are modelled as Transport Services registered to the OS models [3]. The control and the functionalities of the MusicVideoServerSN (which consists of a single process) are shown in Figure 10. The non-funcitonal property .i.e. FrameRate is assigned the required value (40 Frames/sec) in the model element representing MusicVideoServerSN in Figure 4. This non-functional property is further refined to three non-functional properties from the design perspective .i.e., FrameRetrievalTimeMax, ImageCreationTimeMax and ImageSendingTimeMax. These refined non-functional properties are annotated in the behavioural view to their corresponding functionalities .i.e., Get a Frame, Create Image and Send the Image. The OPENcv [19] library functions .i.e., cvQuerryFrame and cvCreateImage and user-space function SendImage providing these functionalities are mentioned below the name of these functionalities. Each of these non-functional requirements are analysed in the performance simulation phase to check whether the required FrameRate has been achieved. Due to the pipelined nature of the functionalities, each of them has to be performed within 1/40 seconds (to fulfil the required frame rate). The function SendImage is a wrapper around the NoTA BSD API Hsend() function [4].

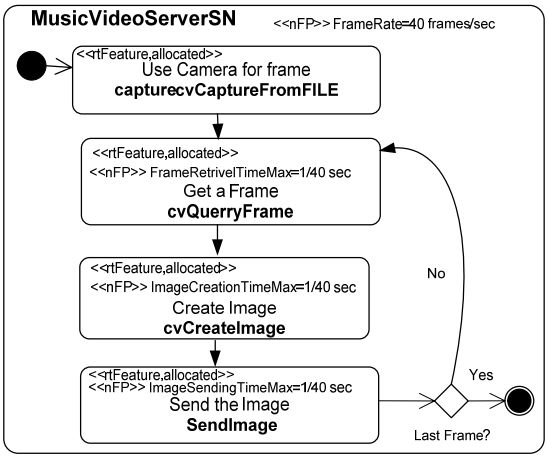

Figure 10: MusicVideoServerSN control with functionalities mentioning refined non-functional properties

Hence a single process of an AN or SN "P_i" can be represented as $P_i = \{C_F, F_1, F_2, \ldots, F_N, S_1, S_2, \ldots S_K\}$, where F_i is a function and S_i is a service requested from platform. The corresponding workload model layer is

$PM_i=\{C_{FM},FM_1,FM_2,\ldots,FM_N, SM_1, SM_2,.. SM_K\}$,
where FM_i is a function workload model and SM_i is a platform service workload model.

C. Extracting Workload Layers

The mapping between the NoTA Application model layers and the corresponding Workload model layers are shown in Table 1.

Table 1: Comparing NoTA application model layers and Workload layers.

Application Layers	Workload Model Layers
use_case= $\{C,N_1,N_2,\ldots,N_N\}$	use_case_model=$\{C_{WLD},NM_1,.,NM_N\}$
$N_i = \{C_P,P_1,P_2,\ldots,P_N\}$	NMi=$\{ C_{PM} ,PM_1,,..,PM_N\}$
$P_i=\{C_F,F_1,F_2,\ldots,F_N,S_1,.. S_K\}$	PMi=$\{C_{FM},FM_1,,FM_N,SM_1,..SM_K\}$

The word mapping does not mean the automatic transformation or extraction of workload models from the application models but facilitates the workload modelling via identification of a workload element corresponding to a certain application model element.

VI. Performance Evaluation and Results

In the case of non-distributed applications, the overall performance model consists of a single platform model to which one or more applications workload models are mapped. These application models represent the processing load of the whole use-case [6]. In case of distributed applications, each server and client (called SNs and ANs in NoTA) in real use-case is modelled as a separate application-level workload model. Each Application-Level workload model of a NoTA AN or SN instantiates the process workload model mimicking its execution in the real use-case. In case of performance models of distributed applications, at least two process workload models are hosted on different platform instances. The processes models hosted on different platform instances communicate with other process models via transport layer OS_Services [3].

In cases where more than one process workload is hosted on same platform, they communicate via inter-process communication (IPC) model of ABSOLUT and load the same platform [18].The performance results for all the platform models are obtained separately and analysed to perform optimizations if the non-functional properties are not satisfied.

A. ABSOLUT Performance Model

In the case study, each NoTA AN and SN presented in the application model is mapped to a separate multi-core based platform model to analyse the performance results and identify the potential bottlenecks at the software and hardware side. The overall performance simulation model is shown in Figure 11. The arrows connecting the Nodes shows the data was transferred between them while the direction of arrows shows the direction of data transfer.

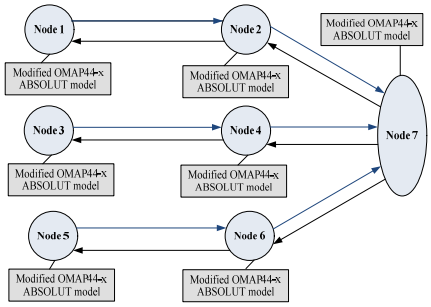

Figure 11: Performance model of the Restaurant Multimdia application

Node 1 and Node 2 represent the NewsServerSN and NewsClientAN. Node 3 and Node 4 represents the MovieStreamerSN and MovieClientAN whereas Node 5 and Node 6 represent the MusicVideoServerSN and MusicVideoClientAN. Node 7 represents the application which is in the form of a control [6].

B. ABSOLUT Platform Model

Each ABSOLUT platform model used in the case study is a modified OMAP-44x platform model. It consists of two ARM Cortex-A9 processors consisting of four cores respectively instead of two (as in case of original TI OMAP44-x platform [20]) along with SDRAM, a POWERVR SGX40 graphics accelerator and an Image signal processor. This is shown in Figure 12. The NoC infrastructure was abstracted out and replaced with on-chip bus as shown in Figure 12.

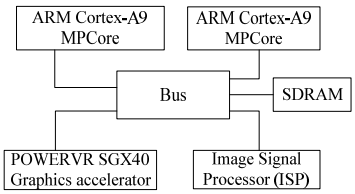

Figure 12: OMAP 44x Platform ABSOLUT model.

Each processor core (Cortex-A9 CPU model) has an L1 and L2 cache and can possibly share an L3 cache with one or more cores in the Multi-Core Processor model. This is shown in Figure 13.

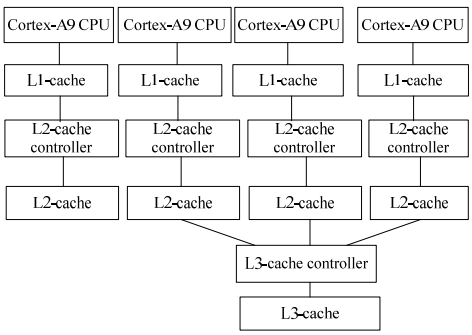

Figure 13: Diagram showing the quad-core processor model used in the performance

C. Application workload Model

All the Servers elaborated in the application model were programmed using OpenCV library [19]. The tool used for the workload extraction is ABSINTH-2 [17] which automates the workload generation of external libraries. The experiment was repeated with different models of

NoTA DIP .i.e., as an OS_Service, Daemon and External library functions.

D. Simulation Scenario (channel bit rate and MAC parameters)

The simulations are carried out in WLAN environment and the simulation parameters for physical and MAC layer are adjusted accordingly as shown in Table 2. These parameters include the IEEE 802.11 DCF configuration parameters and the channel bit rate e.t.c., as explained in [6].

Table 2: Experiment parameters

Parameters	**Values**
SIFS	10 micro seconds
DIFS	50 micro seconds
Slot Interval	20 micro seconds
Preamble Length	144 bits
PLCP header Length	48 bits
Channel bit rate	2 Mbps
CWmin	32
CWmax	2048
CWo	32
EW	16

E. Co-Simulation and performance results

During the execution of application, the end-user requests the bench occupancy and the video of the occupants. The video frames are streamed form the MusicVideoClientAN to the mobile device of the Personal mobile device of a customer. Then the customer invokes other ANs one by one, switching between them after 1→2 minutes each, the ANs then invoke the corresponding SNs to provide the required services to the application.

Each AN and SN workload model is mapped to its respective platforms as shown in Figure 11 and the resultant performance model is run to obtain performance results. The results of all the platforms and their hosted ANs and SNs are written to one text file in the form of different sections, one for each platform and its hosted ANs or SNs. We only present the performance results of the platform hosting the MusicVideoServerSN.

The simulation execution can be easily exited after any pre-decided simulation time, for example after 20 seconds (time in terms of SystemC time model) or another event in the simulation for example the number of streamed packets from one SN to an AN or from a AN to the Application. When the pre-decided condition is met during simulation, the sc_stop() function is called. After that the destructor of the results reporting class is called which writes the gathered simulation results to a text file for analysis.

1) Performance Results (Platform)

Since the MusicVideoServerSN was implemented entirely as software, the Graphics Accelerator and Image Processor Services available from the platform were not used. Therefore only the utilization of the processor cores of platform hosting MusicVideoServerSN is shown in Figure 14. The simulation was run for streaming of 10, 100 and 1000 packets. The solid bar corresponds to 10 packets, bar with horizontal pattern shows use-case of 100 packets and diagonal pattern corresponds to 1000 packets.

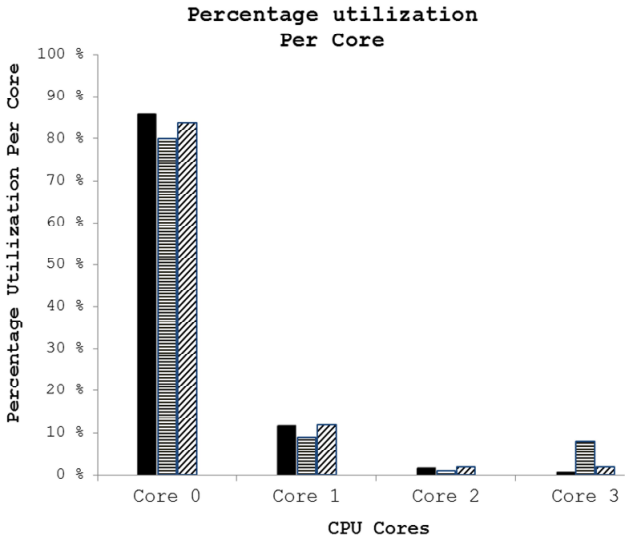

Figure 14: Utilization time of processor cores as compared to overall Utilization time of the CPU

The cache statistics of the platform hosting MusicVideo-ServerSN are shown below for 1000 video frame trans-missions.

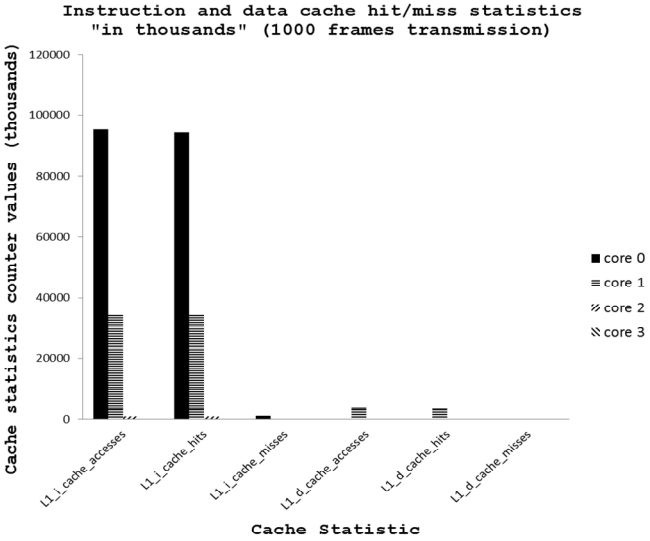

Figure 15: Cache hits miss statistics of the platform hosting MusicVideoServerSN

The performance statistics related to the platform services (MAC and transport protocol models) are recorded via probes. The performance statistics of transport (UDP) and MAC (IEEE 802.11 DCF) layer protocol models (each modelled as an ABSOLUT OS_Service) are shown in Table 3.

Table 3: MAC and Transport Performance statistics

MAC/Transport Performance statistic	**Values**
Throughput at MAC-Level (ratio of successful Frame transmissions and total Frame transmissions)	.99
Throughput at Transport-Level (ratio of successful Packet transmissions and total Packet transmissions)	.98
Average Frame Delays	.52 milliseconds
Average Transport Delays	1.7 milliseconds
Frame loss rate(Percent)	.022%
Packet Loss rate(Percent)	.983%

The results in Table 3 satisfy the non-functional property (FrameRate) only if all the functions in the MusicVideoServerSN which make use of these OS_Services satisfy the non-functional properties from the design perspective. In case of MusicVideoServerSN only the SendImage function makes use of NoTA API function (Hsend() for sending image data) which in turn uses the transport and MAC layer ABSOLUT protocols. As shown in Figure 10, this function must be executed within 25 milliseconds (1/40 seconds) in order to satisfy the required FrameRate of 40 Frames/Second. The processing time of this function along with the other application functions are presented next.

2) Performance Results (Application). Validating Non-Functional Properties

By analysing the processing times of the application source code and the percentage utilization of multi-core processor model by different external library and user-space code, we can find the potential bottlenecks in the application implementation, which will help to perform required optimizations. In other words, after identifying the functionalities which can affect a particular non-functional property, the processing times of these functionalities are analysed to find out whether the implementation of the software components satisfies this non-functional property.

We now elaborate the way the non-functional property the FrameRate is analysed and validated by the performance simulation results. This non-functional property is annotated in the application syntactical view and refined to three non-functional properties in the extended behavioural view as shown in Figure 10. It is shown that due to the pipelined nature of the execution of these functionalities, each of these functionalities must be executed within 1/40 seconds (25 milliseconds) in order to achieve a frame rate of 40 frames/seconds. These functionalities and their corresponding Functions are shown in Table 4

Table 4: Shortlisted functions that can affect the Frame rate (a non-functional property) of FaceTrackerStreamerServer

Functionality	Shortlisted Function
Get a frame from Selected File	cvQuerryFrame
Create Image from Frame	cvCreateImage
Send the Image	SendImage

The processing times and the percentage processor utilization of the aforementioned functions are shown in Figure 16 and Figure 17**.** It is seen that all the operations are performed within 12 milliseconds. The results show that the SendImage function takes less than 6 milliseconds which is well below 25 milliseconds required to achieve the required FrameRate. In this way the results presented in Table 3 are also validated. In other words the performance of MAC and transport protocols is sufficient to satisfy the use case.

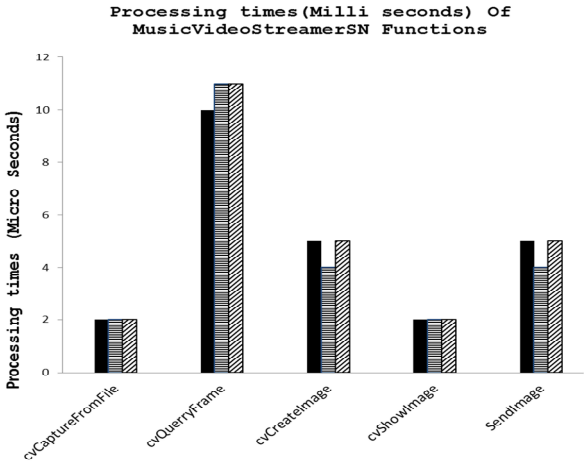

Figure 16: Execution times of functionalities attributing to Frame rate in Face Tracker Subsystem

The processor utilization graph shows that cvQuerryFrame which fetches the a frame for sending to corresponding AN takes 54% of the CPU time in proportion to the overall CPU time taken by all the application functions considered.

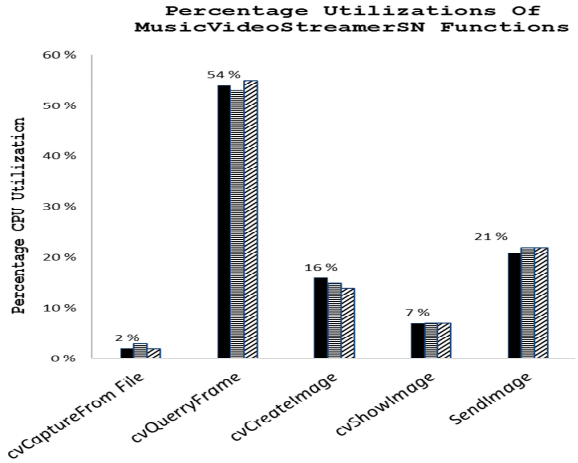

Figure 17: Execution times of functionalities attributing to Frame rate in Face Tracker Streamer Subsystem

The use-case was executed with NoTA DIP operating as an external library and as Platform service. The performance of NoTA DIP as compared to the overall cost of the applications is less the .0001%.These results are confirmed by Vallgrind [13]. The additional delays due to NoTA DIP varied between .9 to 5% of the average transport end-to-end delays.

The obtained performance results are used to perform appropriate changes in the application models by replacing the software components with more light weight implementations or by making changes in the platform model if the performance requirements (non-functional properties) are not met. If the performance requirements are met by all the platform and software components, the architectural exploration stops and the implementation phase starts.

VII. CONCLUSION AND FUTURE WORK

The NoTA application architecture modelling methodology was extended for linking it with the ABSOLUT workload modelling approach used in

performance evaluation. The behavioural view of the NoTA application architecture was extended to obtain a layered application model that can be mapped to the workload model layers used in the applied performance simulation approach. This reduces the time and effort in the performance evaluation phase. The approach was experimented in a RestaurantMultimedia Application case study, UML2.0 MARTE profile as the modelling language and Papyrus toolset.

The use-case was run with NoTA DIP operating as an external library and as Platform service. The performance of NoTA DIP were less than .0001% as compared to the overall performance cost of the application, confirming that NoTA DIP does not act a performance bottleneck and is an efficient service-level interoperability solution.

ACKNOWLEDGEMENTS

This work was performed the Artemis SOFIA project partially funded by TEKES and the European Union. The authors would like to thank also their colleagues for valuable comments and inspiring discussions during the work.

References

[1] Lappeteläinen, Antti et. al., 'Networked Systems, Services, and Information - The Ultimate Digital Convergence'. 1st International Conference on Network on Terminal Architecture, June 11, 2008, Helsinki.

[2] Jari Kreku, Mika Hoppari, Tuomo Kestila, Yang Qu, Juha-Pekka Soininen and Kari Tiensyrja, Combining UML2 and SystemC Application Platform Modelling for Performance Evaluation of Real-Time Systems, EURASIP Journal on Embedded Systems, volume 2008, ARTICLE ID 712329.

[3] Subayal Khan, Jukka Saastamoinen, Mikko Majanen, Jyrki Huusko and Jari Nurmi. Analyzing Transport and MAC Layer in System-Level Performance Simulation, International Symposium on System-on-Chip 2011,Tampere, Finland, October 31 - November 2, 2011.

[4] http://projects.developer.nokia.com/NoTA

[5] B. Kienhuis, E. Deprettere, K. Vissers and P. van der Wolf. Approach for quantitative analysis of application-specific dataflow architectures. The IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP ’97), pp. 338–349, Zurich, Switzerland. July 1997.

[6] Subayal Khan, Susanna Pantsar-Syväniemi, Jari Kreku, Kari Tiensyrjä and Juha-Pekka Soininen, "Linking GENESYS Application Architecture Modelling with Platform Performance Simulation", in Proceedings of the 12th Forum on Specification and Design Languages (FDL 2009), September 22-24, Sophia Antipolis, France, 2009.

[7] USC Center for Software Engineering, Guidelines for Model-Based (System) Architecting and Software Engineering, http://sunset.usc.edu/research/MBASE, 2003.

[8] D. Garlan, R. T. Monroe and D. Wile. Acme: An Architecture Description Interchange Language. Proceedings of CASCON '97, November 1997.

[9] R. Roshande, B. Schmerl, N. Medvidovic and D. Garlan, D. Zhang. Understanding tradeoffs among different architectural modeling approaches. Software Architecture, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference on software architecture.12-15 June 2004, pp. 47 – 56

[10] P. Lieverse, P. van der Wolf, K. Vissers, and E. Deprettere, A methodology for architecture exploration of heterogeneous signal processing systems. Kluwer Journal of VLSI Signal Processing 29 (3), 2001, pp. 197-207.

[11] A. Pimentel and C. Erbas. A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels. IEEE Transactions on Computers, vol. 55, no. 2, Feb. 2006, pp.99 – 112.

[12] T. Wild, A. Herkersdorf and G.-Y. Lee. TAPES—Trace-based architecture performance evaluation with SystemC. Design Automation for Embedded Systems, Vol. 10, Numbers 2-3, Special Issue on SystemC-based System Modeling, Verification and Synthesis, 2006, pp 157-179.

[13] Instantiation and feasibility evaluation of NoTA SOAD via MARTE profile and binary instrumentation. Khan, Subayal; Tiensyrjä, Kari; Nurmi, Jari. The 7th International Conference and Expo on Emerging Technologies for a Smarter World. CEWIT 2010. Incheon, 27 - 29 Sep. 2010.

[14] http://www.isi.edu/nsnam/ns/

[15] http://www.omnetpp.org/

[16] http://valgrind.org/

[17] Application workload model generation methodologies for system-level design exploration. DASIP 2011. November 2-4 2011.Tampere, Finland. Jukka Saastamoinen, Jari Kreku..

[18] Multi-threading support for system-level performance simulation of multi-core architectures. ARCS 2011 - 24th International Conference on Architecture of Computing Systems.Como, Italy. 02/22/2011 - 02/23/2011. Saastamoinen, Jukka; Khan, Subayal; Tiensyrjä, Kari.

[19] http://opencv.willowgarage.com/wiki/

[20] http://www.ti.com/omap

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland

ISBN 978-952-15-2969-6
ISSN 1459-2045